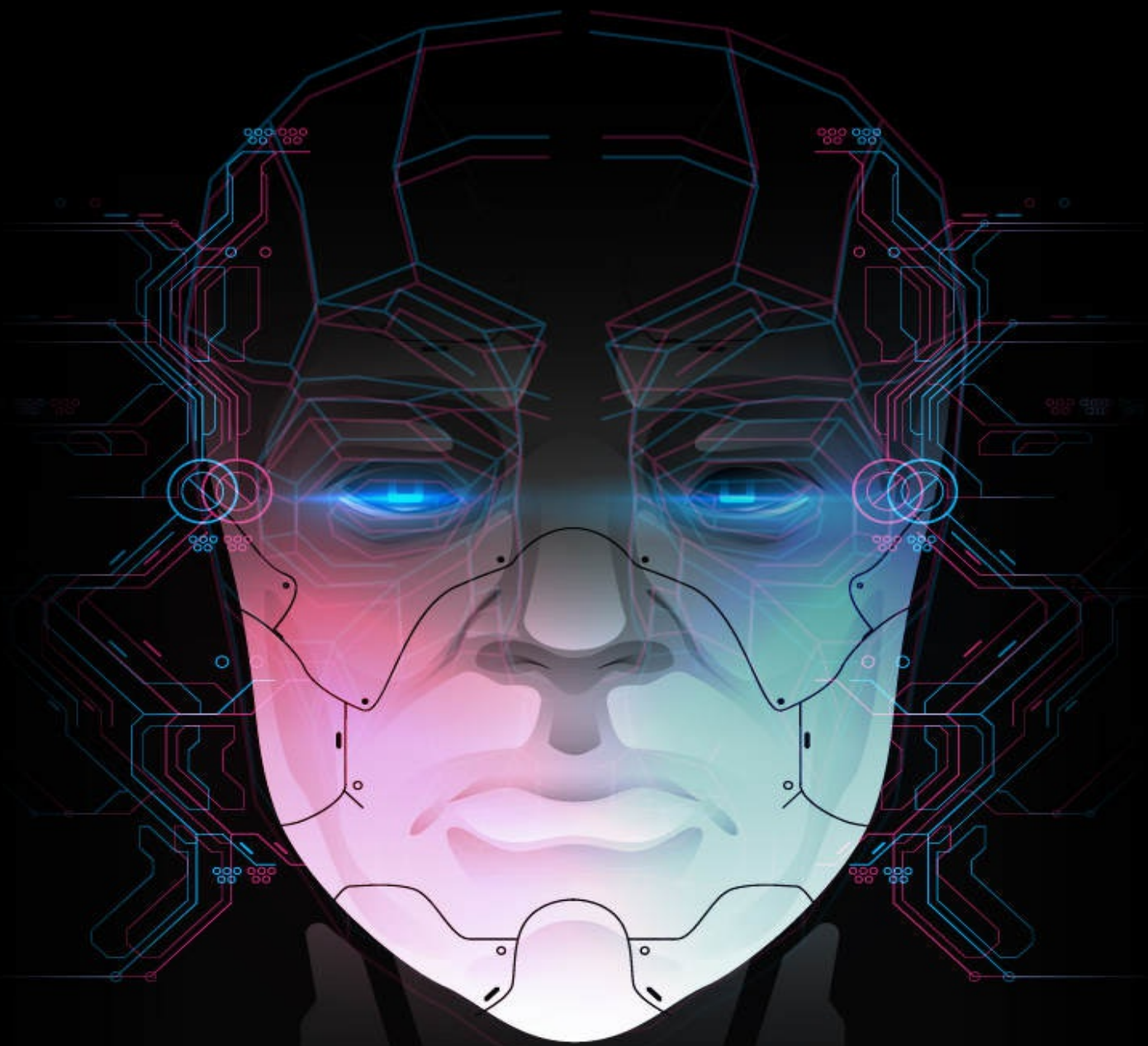
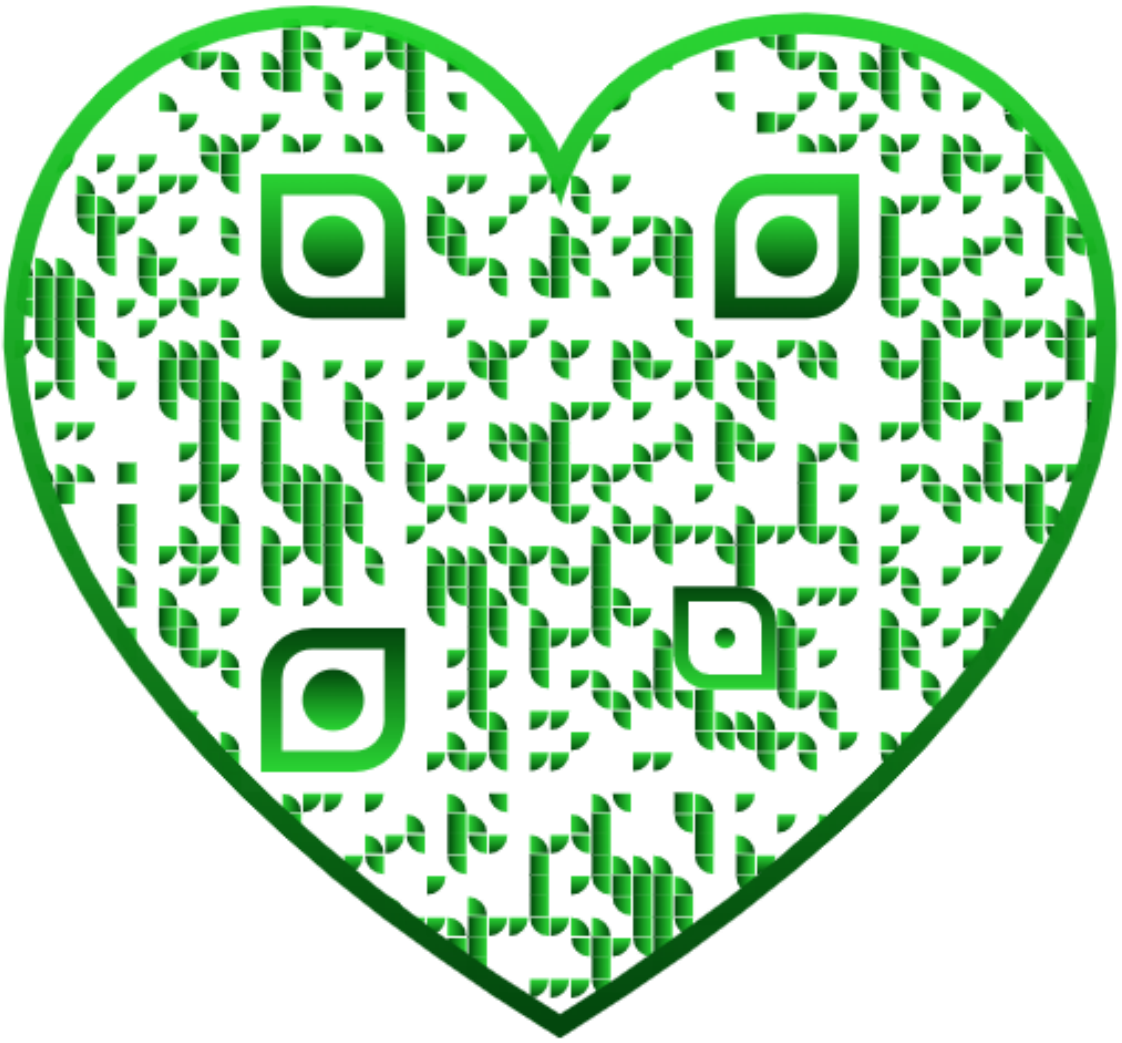
Raghavendra
Yadav



ARTIFICIAL INTELLIGENCE

Crash Course



ARTIFICIAL INTELLIGENCE

Crash Course

Raghavendra Yadav

CONTENT

INTRODUCTION

WHAT IS AI?

SUPERVISED LEARNING

NEURAL NETWORKS AND DEEP LEARNING

TRAINING NEURAL NETWORKS

UNSUPERVISED LEARNING

NATURAL LANGUAGE PROCESSING

REINFORCEMENT LEARNING

SYMBOLIC AI

ROBOTICS

AI PLAYING GAMES

HUMANS AND AI WORKING TOGETHER

WEB SEARCH

ALGORITHMIC BIAS AND FAIRNESS

FUTURE OF AI

GLOSSARY

Introduction

Artificial Intelligence (AI). It’s everywhere, whether we notice it or not. It’s taking the critical decisions in almost every field, from finance to health. It’s driving the algorithms of probably every technology giant’s like Google, Facebook, Amazon, Netflix and many more organizations around the world. And yeah its what puts the “Smart” in front of our devices as we call them smart devices. Artificially Intelligence systems are analysing huge, I mean like huge amounts of data that we’re generating every minute through our smart devices to infer patterns and predicts likely futures. Like the number of drinks that people might buy next week at the store, so the store knows how much to stock. Everyone seems to be talking about AI and how it’s going to revolutionize the world or create some dystopian future. While we don’t know what the future holds, we in this book believe that creating a better outcome starts with a better understanding. The goal of this book is to teach the logic behind AI by tracing its history, examining how it’s being used today, and showing some practical examples. Along the way, we’ll spend time on a hot area of AI known as machine learning. We’ve taught machines to recognize people in photos, predict the next word we’re going to type, and recommend videos for us on YouTube! But AI is broader than just machine learning. So, we’ll talk about how AI systems can act based on plans, rules, and logical representations of the world. These are the algorithms to determine the best move in game of chess or master the most complicated board game “Go”.

AI is still a new field, so there are a lot of problems to solve but not that many people trying to solve them. We also have to think about what the future of AI means for us as humans, & I believe it’s in our best interest to have as many people as possible who understand the field of AI.

Here I am going to introduce my AI robot friend, *Airor* . From now he is going to take on and take you through the crash course of the newly emerging

field Artificial Intelligence.

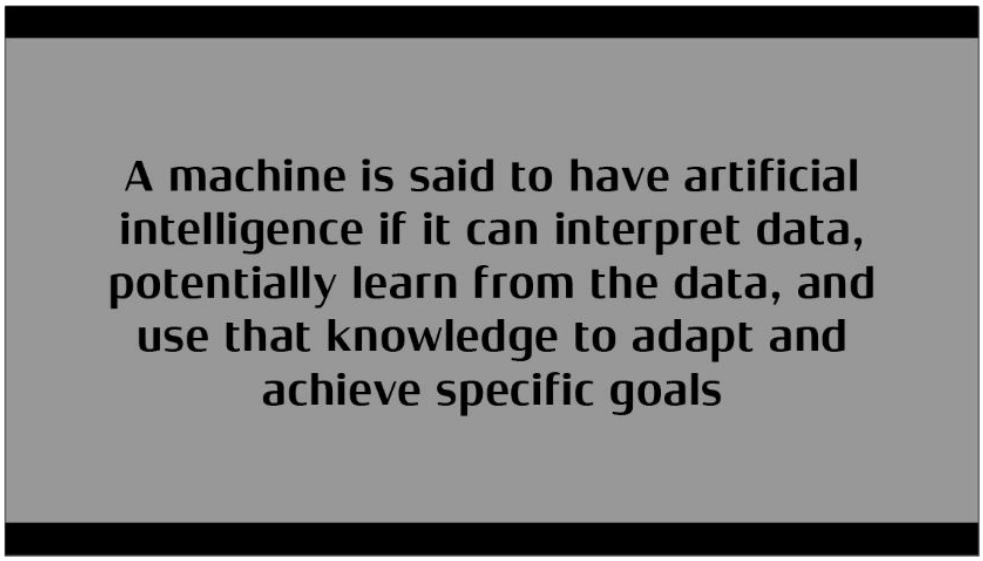
Airor : Hello! Humans. Get on the ride!

What is AI?

Artificial intelligence is everywhere. It's helping banks make loan decisions, and helping doctor's patients, it's on our cell phones, autocompleting texts, it's the algorithm recommending videos to watch. AI already has a pretty huge impact on all of our lives. So people, understandably, have some polarized feelings about it. Some of us imagine that AI will change the world in positive ways, it could end car accidents because we have self-driving cars, or it could give the elderly great, personalized care. While others worry that AI will lead to constant surveillance by a Big Brother government. Some say that automation will take all our jobs or the robots might try and kill us all.

But when we interact with AI that's currently available like Siri, Alexa and many others, it's clear that those are still distant futures. Now to understand where artificial intelligence might be headed, and our role in the AI revolution, we have to understand how we got to where we are today.

If you know about artificial intelligence mostly from movies or fiction books, AI probably seems like this vague label for any machine that can think like a human. Fiction writers like to imagine a more generalized AI, one that can answer any question we might have, and do anything a human can do. But that's a pretty rigid way to think about AI and it's not super realistic.

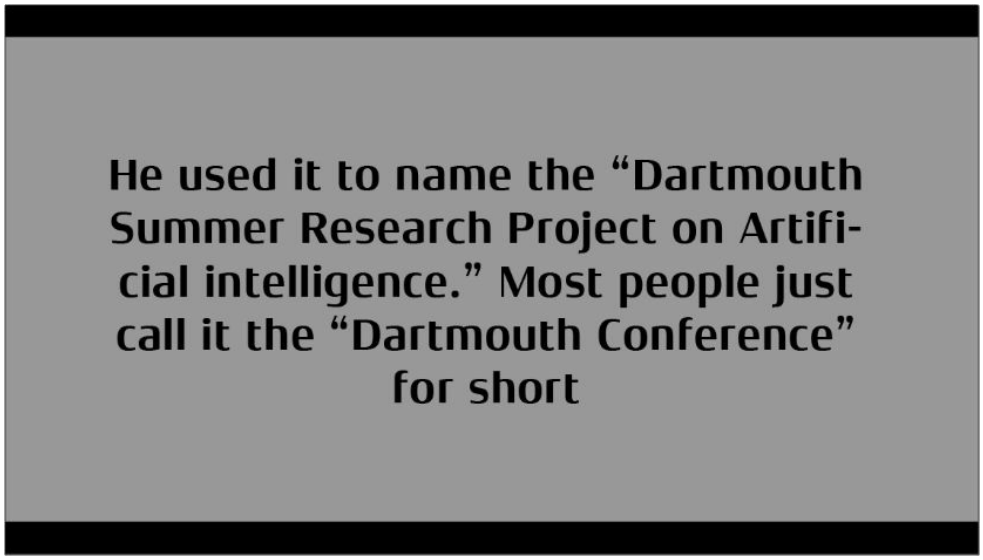

A machine is said to have artificial intelligence if it can interpret data, potentially learn from the data, and use that knowledge to adapt and achieve specific goals

Now, the idea of “learning from the data” is kind of a new approach. But we’ll get into that later. So, let’s say we load up a new program in our robot *Airor* . This program looks at a bunch of photos, some of me and some of not me, and then learns from those data. Then, we can show him a new photo of mine and he can recognize that the photo is me. If he can correctly classify that new photo, we could say that *Airor* has some artificial intelligence! Of course, that’s a very specific input of photos, and a very specific task of classifying a photo that’s either me or not me. With just that facial recognition program *Airor* can’t recognize or name anyone who isn’t me. He can’t navigate to places or hold any meaningful conversations.

Airor can’t do most things that humans do, which is pretty standard for AI these days. But even with this much more limited definition of artificial intelligence, AI still plays a huge role in our everyday lives. There are some more obvious uses of AI, like Alexa or Roomba, which is kind of like the AI from the science fiction I guess. But there are a ton of less obvious examples. When we buy something in a big store or online, we have one type of AI deciding which and how many items to stock. And as we scroll through Instagram, a different type of AI picks ads to show us. AI helps determine how expensive our car insurance is, or whether we get approved for a loan. And AI even affects big life decisions. Like when you submit your college or job application AI might be screening it before a human even sees it.

The way AI and automation is changing everything, from commerce to

jobs, is sort of like the Industrial Revolution in the 18Th century. This change is global, some people are excited about it, and others are afraid of it. But either way, we all have the responsibility to understand AI and figure out what role AI will play in our lives. The AI revolution itself isn't even that old. The term artificial intelligence didn't even exist a century ago. It was coined in 1956 by a computer scientist named John McCarthy.


He used it to name the "Dartmouth Summer Research Project on Artificial intelligence." Most people just call it the "Dartmouth Conference" for short

The Dartmouth conference lasted eight weeks and got a bunch of computer scientists, cognitive psychologists, and mathematicians to join the conference. Many concepts that we'll talk about later, like artificial neural networks, were dreamed up and developed during this conference and in the few years that followed. But because these excited academics were really optimistic about artificial intelligence, they may have oversold it a bit.

For example, Marvin Minsky was a talented cognitive scientist who was part of the Dartmouth Conference. But he also had some ridiculously wrong predictions about technology, and specifically AI. In 1970, he claimed that in "Three to eight years we will have a machine with the general intelligence of an average human being." We're not even close to that today. Scientists at the

Dartmouth Conference seriously underestimated how much data and computing power an AI would need to solve complex, real world problems.

Artificial Intelligence doesn't really "know" anything when it's first created, kind of like a human baby. Babies use their senses to perceive the world and their bodies to interact with it, and they learn from the consequences of their actions. My baby niece might put a strawberry in her mouth and decide that it's tasty and then she might put play dough in her mouth and decide that it's gross. Babies experience millions of these data gathering events as they learn to speak, walk, think.

Now, most kinds of artificial intelligence don't have things like senses, a body, or a brain that can automatically judge a lot of different things like a human body does. Modern AI systems are just programs in machines. So, we need to give AI a lot of data. Plus, we have to label the data with whatever information the AI is trying to learn, like whether food tastes good to humans. And then, the AI needs a powerful enough computer to make sense of all the data. All of this wasn't available in 1956. Back then, an AI could maybe tell the difference between a triangle and a circle, but it definitely couldn't recognize somebodies face like todays AI does.

Until about 2010 or so, the field was basically frozen in what's called the **AI Winter** . Still there were a lot of changes in the last half a century that led us to the AI Revolution. The AI Revolution didn't begin with a single event, idea, or invention. We got to where we are today because of lots of small decisions, and two big developments in computing. The first development was a huge increase in computing power and how fast computers could process data.

During the Dartmouth Conference in 1956, the most advanced computer was the IBM 7090. It filled a whole room, stored data on basically giant cassette tapes, and took instructions using paper punch cards. Every second, the IBM 7090 could do about 200,000 operations. But if you tried to do that it would take you 55 and a half hours! Assuming you did one operation per second, and took no breaks. That's right. Not. Even. For. Snacks.

At the time, that was enough computing power to help with the U.S. Air Force's Ballistic Missile Warning System. But AI needs to do a lot more computations with a lot more data. The speed of a computer is linked to the number of transistors it has to do operations. Every two years or so since

1956, engineers have doubled the number of transistors that can fit in the same amount of space. So, computers have gotten much faster.

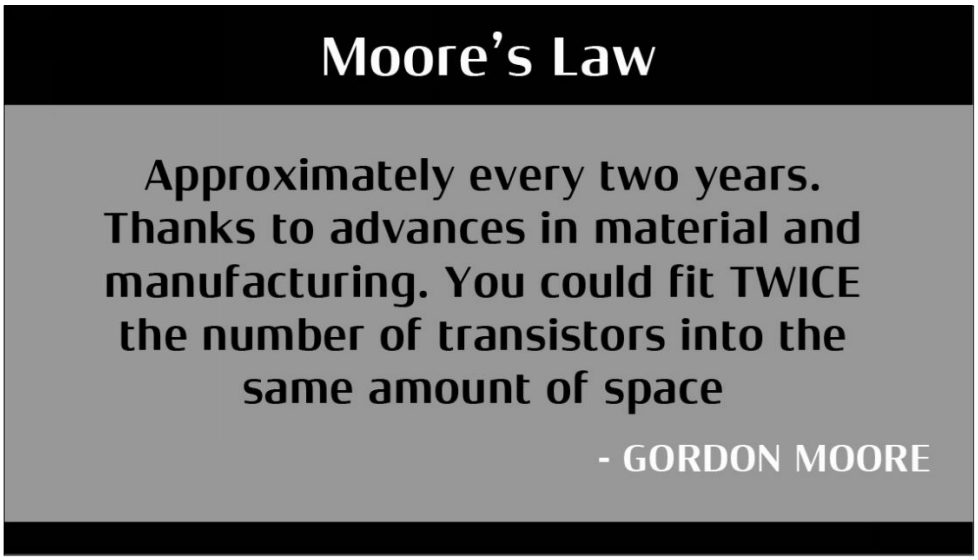
When the first iPhone was released in 2007, it could do about 400 million operations per second. But today, Apple says the iPhone 12's processor A14 Bionic chipset can do about 11 trillion operations per second. That's like having the computing power of over a thousand original iPhones (1st generation) in your pocket. (For all the nerds out there, listen you're right, it's not quite that simple - we're just talking about FLOPS here) And a modern supercomputer, which does computational functions like the IBM 7090 did, can do over 442 quadrillion operations per second. To put it another way, a program that would take a modern supercomputer one second to compute, would have taken the IBM 7090 4,753 years.

Computers started to have enough computing power to mimic certain brain functions with artificial intelligence around 2005, and that's when the AI winter started to show signs of thawing. But it doesn't really matter if you have a powerful computer unless you also have a lot of data for it to munch on. The second development that kicked off the AI revolution is something that you're using right now: The Internet and social media. In the past 20 years, our world has become much more interconnected. Whether you livestream from your phone, or just use a credit card, we're all participating in the modern world. Every time we upload a photo, click a link, tweet a hashtag, tweet without a hashtag, like a YouTube video, tag a friend on Facebook, argue on Reddit, post on TikTok, support a Kickstarter campaign, buy products on Amazon, call an Uber from a party, and basically ANYTHING, that generates data.

Even when we do something that seems like it's offline, like applying for a loan to buy a new car or using a passport at the airport those datasets end up in a bigger system. The AI revolution is happening now, because we have this wealth of data and the computing power to make sense of it. The idea that we're generating a bunch of data but don't always know how, why, or if it's being used by computer programs can be kind of overwhelming. But through this book, we will learn how artificial intelligence works because it's impacting our lives in huge ways. And that impact will only continue to grow.

With knowledge, we can make small decisions that will help guide the AI

revolution, instead of feeling like we're riding a rollercoaster we didn't sign up for. We're creating the future of artificial intelligence together, every single day.


Moore's Law
Approximately every two years. Thanks to advances in material and manufacturing. You could fit TWICE the number of transistors into the same amount of space
- GORDON MOORE

Supervised Learning

Today, we're going to try to teach our bot Airor something.

Hey Airor!

Airor : "Hello humanoid friend!"

Are you ready to learn?

Airor : "Hello humanoid friend!"

As you can see, he has a lot of learning to do, which is the basic story of all artificial intelligence. But it's also our story. Humans aren't born with many skills, and we need to learn how to sort mail, land airplanes, and have friendly conversations. Computer scientists have tried to help computers learn like we do, with a process called **supervised learning** .

You ready, Airor?

Airor : "Hello humanoid friend!"

The process of learning is how anything can make decisions, like for example humans, animals, or AI systems. They can adapt their behaviour based on their experiences. In this book, we'll talk about three main types of learning:

1. **Reinforcement Learning.**
2. **Unsupervised Learning.**
3. **Supervised Learning.**

Reinforcement Learning is the process of learning in an environment, through feedback from an AI's behaviour, it's how kids learn to walk! No one tells them how, they just practice, stumble, and get better at balancing until they can put one foot in front of the other.

Unsupervised Learning is the process of learning without training labels. It could also be called clustering or grouping. Sites like YouTube use unsupervised learning to find patterns in the frames of a video, and compress

those frames so that videos can be streamed to us quickly.

Supervised Learning is the process of learning with training labels. It's the most widely used kind of learning when it comes to AI, and it's what we'll focus. Supervised learning is when someone who knows the right answers, called a supervisor, points out mistakes during the learning process. You can think of this like when a teacher corrects a student's math.

In one kind of supervised setting, we want an AI to consider some data, like an image of an animal, and classify it with a label, like "reptile" or "mammal." AI needs computing power and data to learn. And that's especially true for supervised learning, which needs a lot of training examples from a supervisor. After training this hypothetical AI, it should be able to correctly classify images it hasn't seen before, like a picture of a kitten as a mammal. That's how we know it's learning instead of just memorizing answers. Supervised learning is a key part of lots of AI you interact with every day! It's how email accounts can correctly classify a message from your boss as important, and ads as spam. It's how Facebook tells your face apart from your friend's face so that it can make tag suggestions when you upload a photo. And it's how your bank may decide whether your loan request is approved or not.

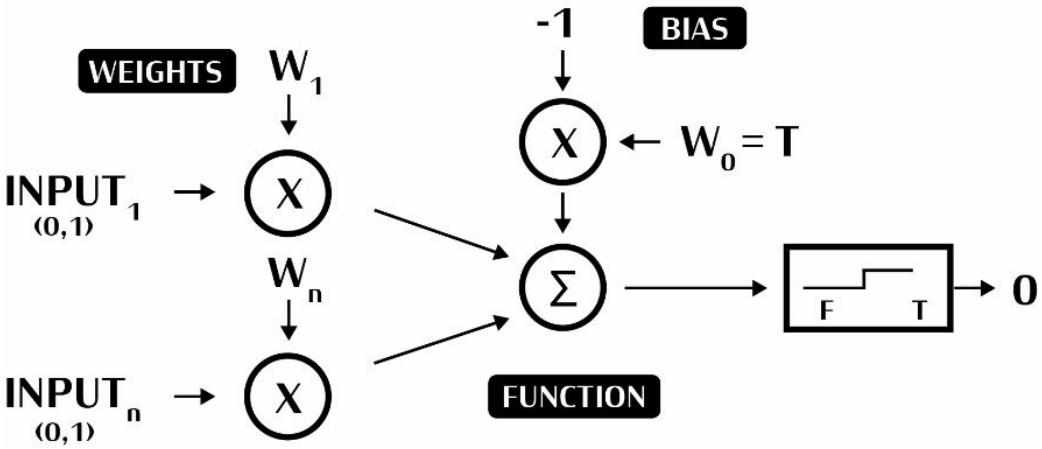
Now, to initially create this kind of AI, computer scientists were loosely inspired by human brains. They were mostly interested in cells called neurons, because our brains have billions of them. Each neuron has three basic parts: the cell body, the dendrites, and the axon. The axon of one neuron is separated from the dendrites of another neuron by a small gap called a synapse. And neurons talk to each other by passing electric signals through synapses. As one neuron receives signals from other neurons, the electric energy inside of its cell body builds up until a threshold is crossed. Then, an electric signal shoots down the axon, and is passed to another neuron where everything repeats.

Now the goal of early computer scientists wasn't to mimic a whole brain. Their goal was to create one artificial neuron that worked like a real one. In 1958, a psychologist named Frank Rosenblatt was inspired by the Dartmouth Conference and was determined to create an artificial neuron. His goal was to teach this AI to classify images as "triangles" or "not-triangles" with his supervision. That's what makes it supervised learning! The machine he built

was about the size of a grand piano, and he called it the Perceptron. Rosenblatt wired the Perceptron to a 400pixel camera, which was hi-tech for the time, but is about a billion times less powerful than the one on the back of your modern Smartphone. He would show the camera a picture of a triangle or a not-triangle, like a circle. Depending on if the camera saw ink or paper in each spot, each pixel would send a different electric signal to the Perceptron. Then, the Perceptron would add up all the signals that match the triangle shape. If the total charge was above its threshold, it would send an electric signal to turn on a light. That was artificial neuron speak for “yes, that’s a triangle!” But if the electric charge was too weak to hit the threshold, it wouldn’t do anything and the light wouldn’t turn on, that meant “not a triangle.”

At first, the Perceptron was basically making random guesses. So, to train it with supervision, Rosenblatt used “yes” and “no” buttons. If the Perceptron was correct, he would push the “yes” button and nothing would change. But if the Perceptron was wrong, he would push the “no” button, which set off a chain of events that adjusted how much electricity crossed the synapses, and adjusted the machine’s threshold levels. So, it would be more likely to get the answer correct next time!

Nowadays, rather than building huge machines with switches and lights, we can use modern computers to program AI to behave like neurons. The basic concepts are pretty much the same: First, the artificial neuron receives inputs multiplied by different weights, which correspond to the strength of each signal. In our brains, the electric signals between neurons are all the same size, but with computers, they can vary. The threshold is represented by a special weight called a bias, which can be adjusted to raise or lower the neuron’s eagerness to fire. All the inputs are multiplied by their respective weights, added together, and a mathematical function gets a result. In the simplest AI systems, this function is called a step function, which only outputs a 0 or a 1. If the sum is less than the bias, then the neuron will output a 0, which could indicate not-triangle or something else depending on the task. But If the sum is greater than the bias, then the neuron will output a 1, which indicates the opposite result!



An AI can be trained to make simple decisions about anything where you have enough data and supervised labels: triangles, junk mail, languages, movie genres, or even similar looking foods. Like donuts and bagels.

Hey *Airor!* You want to learn how to sort some disgusting bagels from delicious donuts?

Airor : "Hello humanoid friend!"

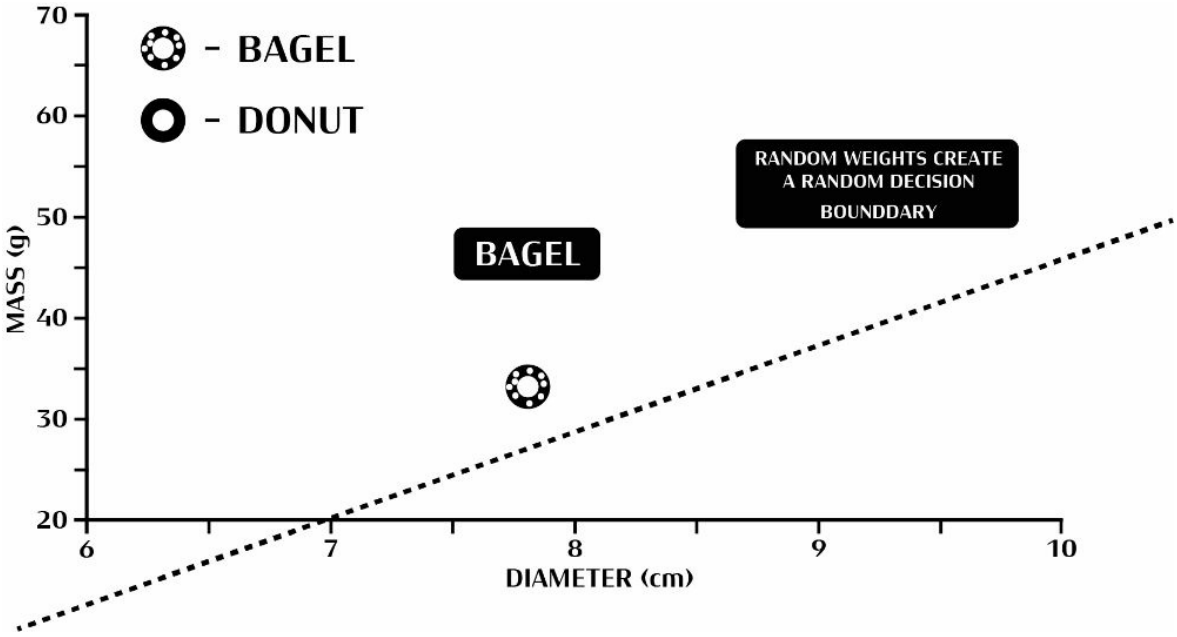
Airor still has the talk-like-a-human program! So, I need to swap this out for a perceptron program.

Now that *Airor* is ready to learn, we'll measure the mass and diameter of some bagels and donuts, and supervise him so he gets better at labelling them.

Right now, he doesn't know anything about bagels or donuts or what their masses and diameters might be. So, his program is initially using random weights for mass, diameter, and the bias to help make a decision. But as he learns, those weights will be updated! Now, we can use different mathematical functions to account for how close or far an AI is from the correct decision, but we're going to keep it simple. *Airor's* perceptron program is using a step function, so it's an either-or choice. 0 or 1. Bagel or donut. Completely right or completely wrong.

First item has a mass of 34 grams and a diameter of 7.8 centimeters. The perceptron takes these inputs (mass and diameter), multiplies them by their respective weights, then adds them together. If the sum is greater than the

bias which, remember, is the threshold for the neuron firing *Airor* will say "bagel." So, if it helps to think of it this way, the bias is like a bagel threshold. If the sum is less than the bias, it hasn't crossed the bagel threshold, and *Airor* will say "donut." All this math can be tricky to picture. So, to visualize what's going on, we can think of *Airor's* perceptron program as a graph, with mass on one axis and diameter on the other. The weights and bias are used to calculate a line called a decision boundary on the graph, which separates bagels from donuts. And if we represent this same item as a data point, we'd graph it at 34 grams and 7.8 centimeters. This data point is above the decision boundary, in the bagel zone!



So, all this means is that when I ask *Airor* what this food is… he'll say

Airor: "Bagel!"

And... he got it wrong, because this is a donut. No big deal! With a brand new program, he's like a baby that made a random guess! Because he's using random weights right now. But we can help him learn by updating his weights. So, we take an old weight and add a number calculated by an equation called the update rule. We're going to keep this conceptual, but if you want more information about this equation, we've linked to a resource in the resources section.

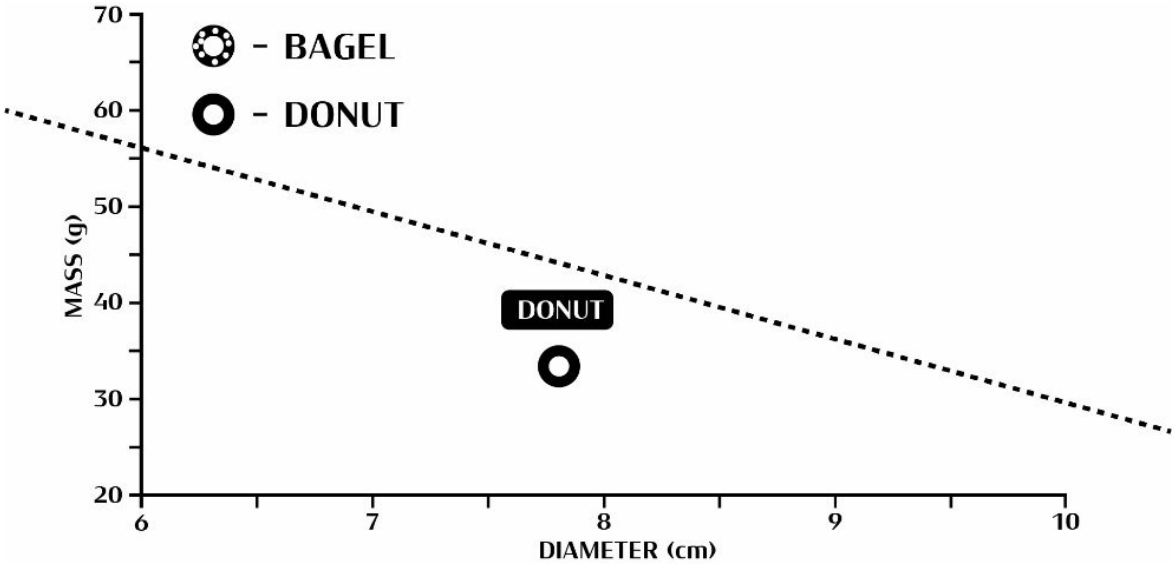
Now because our perceptron can only be completely right or completely wrong, the update rule ends up being pretty simple. If *Airor* made the right choice, like labelling a donut as a donut, the update rule works out to be 0. So, he adds 0 to the weight, and the weight stays the same.

	BAGEL	DONUT
DONUT		*Airor*
UPDATE RULE	0	

But if *Airor* made the wrong choice, like labelling a donut as a bagel, the update rule will have a value a small positive or negative number. He'll add that value to the weight, and the weight will change.

	BAGEL	DONUT
DONUT	*Airor*	
UPDATE RULE	+n OR -n	

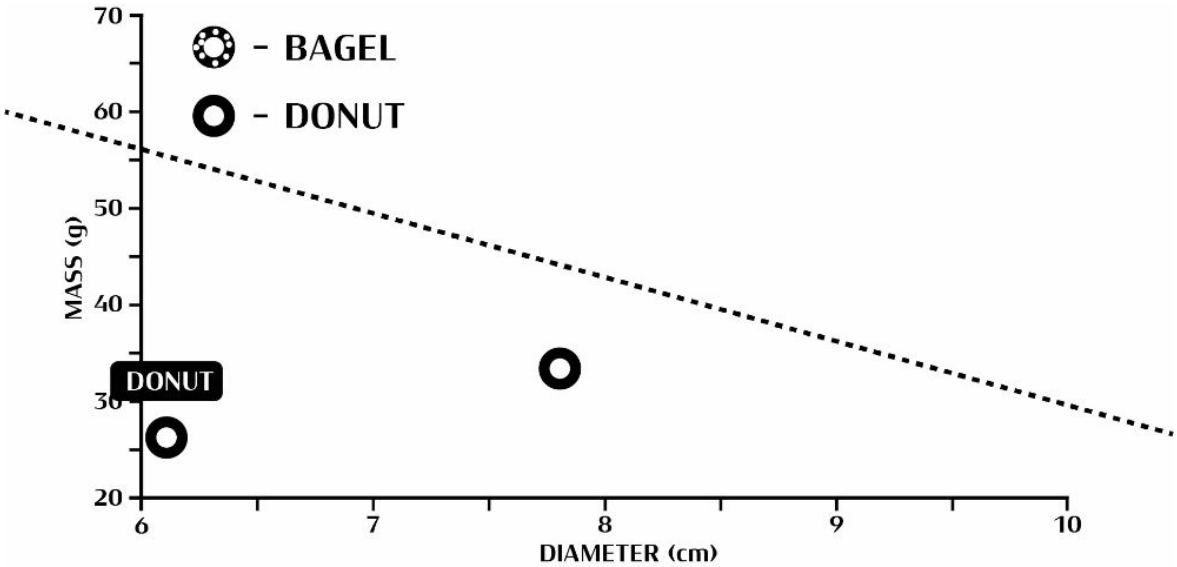
Conceptually, this means *Airor* learns from failure but not from success. So, he called donut a bagel, and got the label wrong. By pressing the "no" button, I'm supervising his learning and letting him know he made the wrong choice. So, his weights update. If we look back at the graph, we can see that when the weights update, the decision boundary changes. The data point we added is now below the line, in the donut zone. Now, his perceptron will classify another item with this mass and diameter as a donut!



Now this next item [donut] has a mass of 26 grams and a diameter of 6.1 centimeters. What do you think, *Airor?*

Airor: "Donut!"

He got it right! When he took those inputs and did that same calculation, the sum was less than the bias. That data point appeared below the decision boundary in the donut zone.

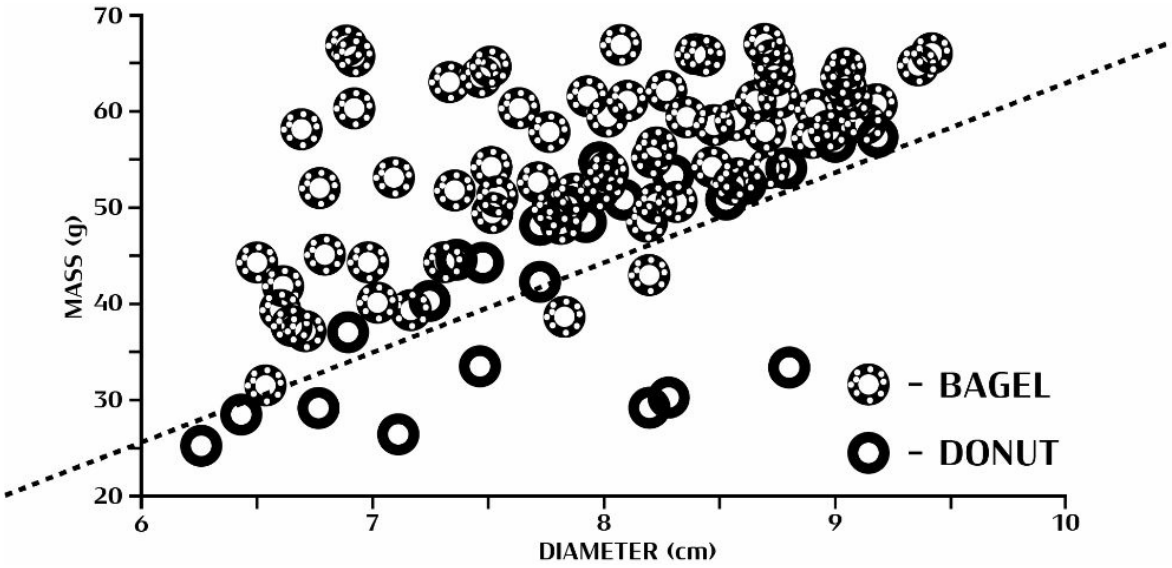


So, I'm going to push the "yes" button. In this case, the update rule equation works out to 0, so the weights stay the same, and so does the

decision boundary. Now we'll show 48 more times to train his perceptron.

After we're done training *Airor's* perceptron, we have to test it on new data to see how well he learned. So, I've got 100 new bagels and donuts for him to classify.

Alright so overall, he classified 25 donuts and 75 bagels. We can visualize the results on the graph with the decision boundary like this.



But we can also put the results in a table, called a confusion matrix, because it tells us where *Airor* was confused.

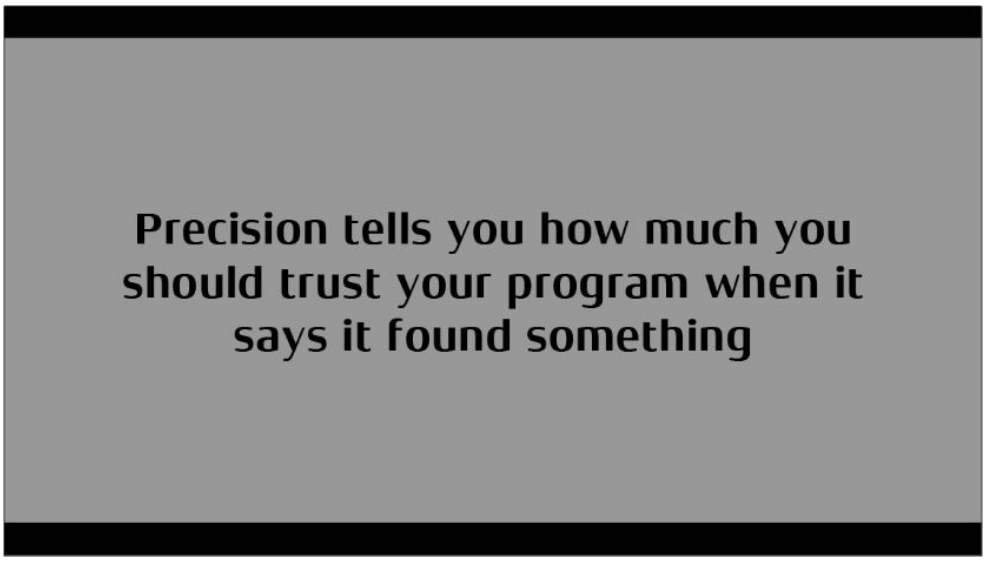
CONFUSION MATRIX

	AIROR SAYS DONUT	AIROR SAYS BAGEL	TOTAL
ACTUALLY DONUT	8	17	25
ACTUALLY BAGEL	2	73	75
TOTAL	10	90	100 TOTAL

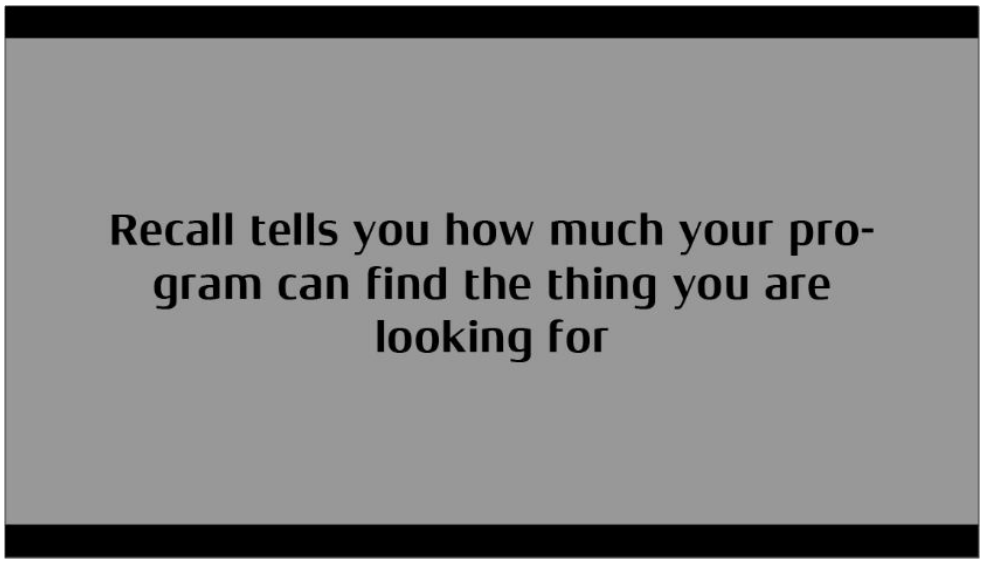
(8 + 73) / 100 = 81%

He got 8 donuts correct and 73 bagels correct. But he said that a bagel was a donut twice, and that a donut was a bagel 17 times. Using these numbers, we can calculate his overall accuracy by adding together what he got right, which were 8 donuts and 73 bagels, and dividing by the total 100, to get 81%.

But to really understand what's wrong, we need to look at his precision and his recall. We can calculate these percentages for both foods, but we'll focus on donuts right now.



If *Airor* tells me something's a donut, I'm expecting to eat a donut. I don't want to bite into a bagel, because that would be a gross surprise. Of the 10 items that he said were donuts, 8 were actually donuts. So, he was 80% precise, and I can be 80% sure he's only handing me donuts when he says he is.



I'm really hungry, so I want as many donuts as possible. But of the 25

items that were donuts, he correctly labelled 8 of them. So, his recall was just 32%, and he just handed me 32% of all the donuts. The precision and recall depend on the criteria *Airor* is using to make a decision: diameter and mass. And as we can see from the above graph, he thinks that donuts generally have smaller diameters and masses than bagels they're small, fluffy treats. So, when it comes to classifying donuts, he has high precision. Because if he says something's a donut, we're pretty sure it's a donut, not a disgusting bagel.

But *Airor* has low recall, because this criteria didn't account for the fact that some donuts can be way bigger than the donuts we used to train his perceptron. They have a bigger diameter and mass, and they fall in the current "bagel zone," so he missed a lot of donuts when he was classifying.

Figuring out what criteria to use is the key to most AI challenges. If we wanted better accuracy for this donut-bagel problem, maybe we should have used inputs besides mass and diameter, like checking for seeds or sprinkles. Generally, more inputs are better for accuracy, but the AI will need more processing power and time to make decisions. An ideal AI system would be small, powerful, and have perfect precision and perfect recall.

But in the real world, mistakes happen, so we have to prioritize based on our goals. The AI filtering our inboxes needs to make sure we get all the important emails, so it needs high recall. But it's okay if it isn't very precise, because we can deal with some spam getting through and don't need only good emails. Most AIs handle more complicated problems than sorting something into one of two categories, though. The world isn't all donuts and bagels. So, to answer more complicated questions, we need more complicated AI. In next chapter, we'll combine artificial neurons to create an artificial neural network.

Neural Networks and Deep Learning

We taught *Airor* to learn using a perceptron, a program that imitates one neuron. But our brains make decisions with 100 billion neurons, which have trillions of connections between them! We can actually do a lot more with AI if we connect a bunch of perceptron's together, to create what's called an artificial neural network.

Neural networks are better than other methods for certain tasks like, image recognition. The secret to their success is their hidden layers, and they're mathematically very elegant. Both of these reasons are why neural networks are one of the most dominant machine learning technologies used today.

Not that long ago, a big challenge in AI was real-world image recognition, like recognizing a dog from a cat, and a car from a plane from a boat. Even though we do it every day, it's really hard for computers. That's because computers are good at literal comparisons, like matching 0s and 1s, one at a time. It's easy for a computer to tell that these images are the same by matching the pixels. But before AI, a computer couldn't tell that these images are of the same dog, and had no hope of telling that all of these different images are dogs. So, a professor named Fei-Fei Li and a group of other machine learning and computer vision researchers wanted to help the research community develop AI that could recognize images. The first step was to create a huge public dataset of labelled real-world photos.

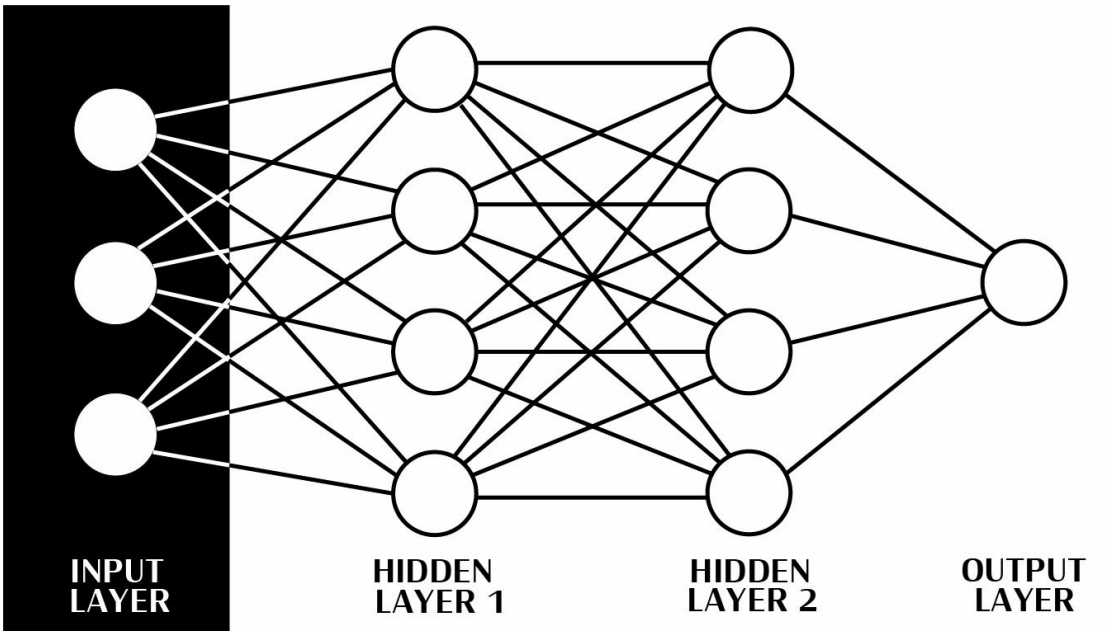
That way, computer scientists around the world could come up with and test different algorithms. They called this dataset ImageNet. It has 3.2 million labelled images, sorted into 5,247 nested categories of nouns. Like for example, the "dog" label is nested under "domestic animal," which is nested under "animal." Humans are the best at reliably labelling data. But if one person did all this labelling, taking 10 seconds per label, without any sleep or snack breaks, it would take them over a year! So ImageNet used crowd-

sourcing and leveraged the power of the Internet to cheaply spread the work between thousands of people. Once the data was in place, the researchers started an annual competition in 2010 to get people to contribute their best solutions to image recognition.

Alex Krizhevsky, who was a graduate student at the University of Toronto. In 2012, he decided to apply a neural network to ImageNet, even though similar solutions hadn't been successful in the past. His neural network, called AlexNet, had a couple of innovations that set it apart. He used a lot of hidden layers. He also used faster computation hardware to handle all the math that neural networks do. AlexNet outperformed the next best approaches by over 10%. It only got 3 out of every 20 images wrong.

Since 2012, neural network solutions have taken over the annual competition, and the results keep getting better and better. Plus, AlexNet sparked an explosion of research into neural networks, which we started to apply to lots of things beyond image recognition.

To understand how neural networks can be used for these classification problems, we have to understand their architecture first. All neural networks are made up of an input layer, an output layer, and any number of hidden layers in between. There are many different arrangements but we'll use the classic multi-layer perceptron as an example. The input layer is where the neural network receives data represented as numbers. Each input neuron represents a single feature, which is some characteristic of the data.



Features are straightforward if you're talking about something that's already a number, like grams of sugar in a donut. But, really, just about anything can be converted to a number. Sounds can be represented as the amplitudes of the sound wave. So, each feature would have a number that represents the amplitude at a moment in time. Words in a paragraph can be represented by how many times each word appears. So, each feature would have the frequency of one word. Or, if we're trying to label an image of a dog, each feature would represent information about a pixel. So, for a grayscale image, each feature would have a number representing how bright a pixel is. But for a color image, we can represent each pixel with three numbers: the amount of red, green, and blue, which can be combined to make any color on your computer screen.

Once the features have data, each one sends its number to every neuron in the next layer, called the hidden layer. Then, each hidden layer neuron mathematically combines all the numbers it gets. The goal is to measure whether the input data has certain components. For an image recognition problem, these components may be a certain color in the centre, a curve near the top, or even whether the image contains eyes, ears, or fur. Instead of answering yes or no, like the simple Perceptron program, each neuron in the hidden layer does some slightly more complicated math and outputs a number. And then, each neuron sends its number to every neuron in the next layer, which could be another hidden layer or the output layer.

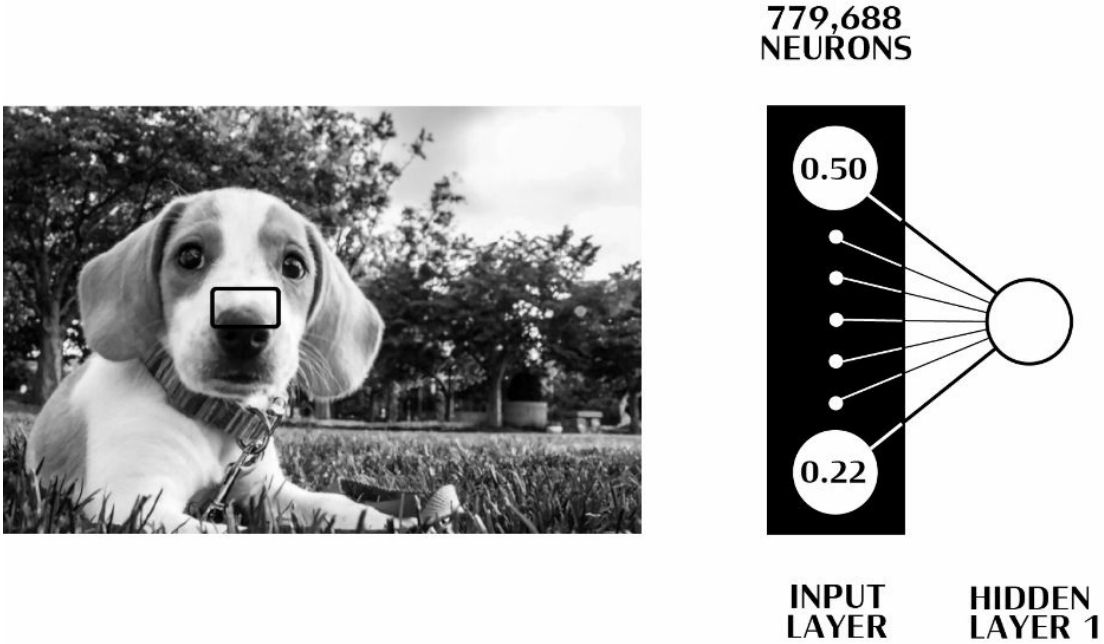
The output layer is where the final hidden layer outputs are mathematically combined to answer the problem. So, let’s say we’re just trying to label an image as a dog. We might have a single output neuron representing a single answer - that the image is of a dog or not. But if there are many answers, like for example if we’re labelling a bunch of images, we’ll need a lot of output neurons. Each output neuron will correspond to the probability for each label like for example, dog, car and more. And then we can pick the answer with the highest probability.

The key to neural networks and really all of AI -- is math. A neural network kind of seems like a black box that does math and spits out an answer. I mean, those middle layers are even called hidden layers! But we can understand the gist of what’s happening by working through an example.

Airor?

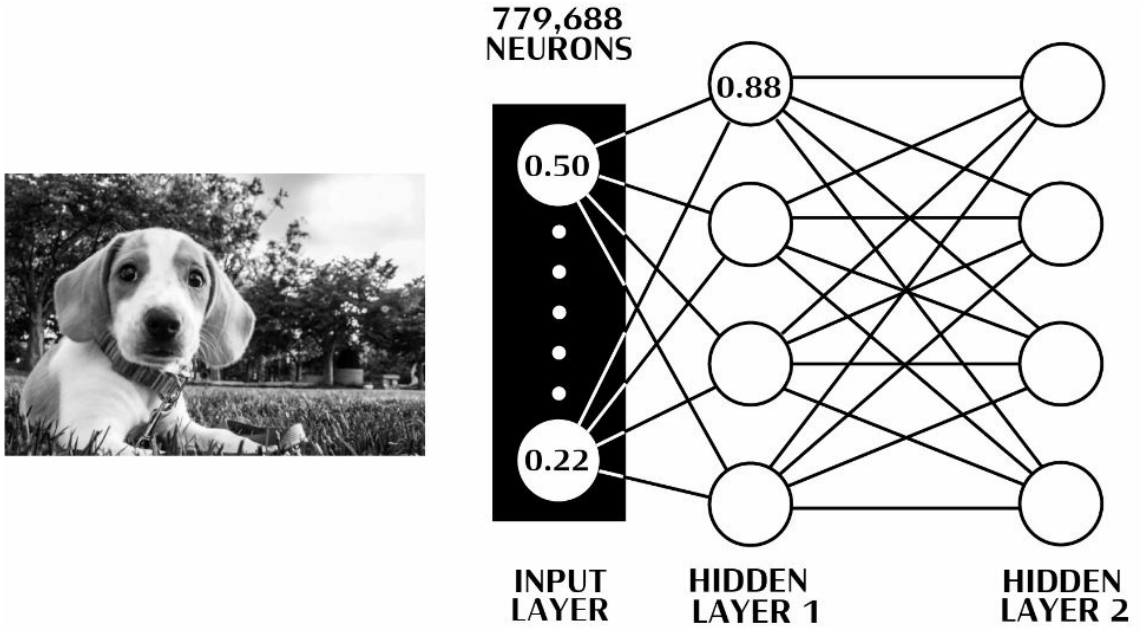
Let’s give *Airor* a program with a neural network that’s been trained to recognize a dog in a grayscale photo.

When we show him this photo first, every feature will contain a number between 0 and 1 corresponding to the brightness of one pixel. And it’ll pass this information to the hidden layer. Now, let’s focus on one hidden layer neuron. Since the neural network is already trained, this neuron has a mathematical formula to look for a particular component in the image, like a specific curve in the centre. The curve at the top of the nose. If this neuron is focused on this specific shape and spot, it may not really care what’s happening everywhere else. So, it would multiply or weigh the pixel values from most of those features by 0 or close to 0.

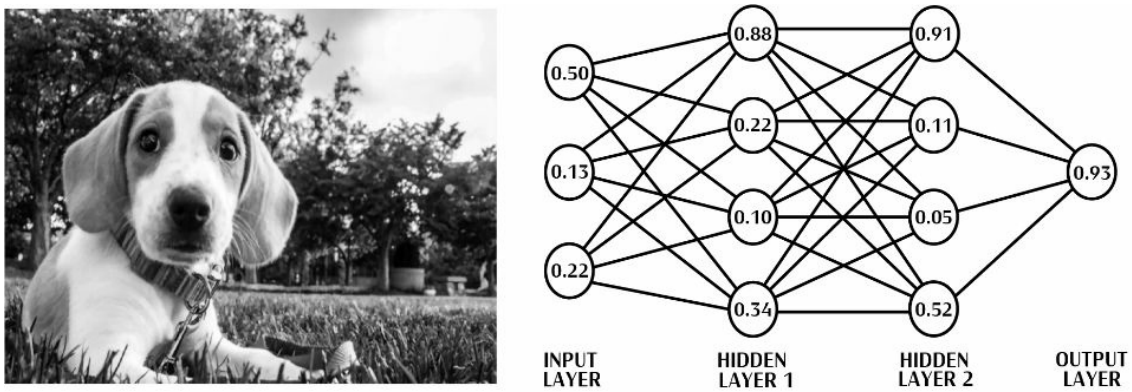


Because it’s looking for bright pixels here, it would multiply these pixel values by a positive weight. But this curve is also defined by a darker part below. So, the neuron would multiply these pixel values by a negative weight. This hidden neuron will add all the weighted pixel values from the input neurons and squish the result so that it’s between 0 and 1. The final number basically represents the guess of this neuron thinking that a specific curve, aka a dog nose, appeared in the image. Other hidden neurons are looking for other components, like for example, a different curve in another part of the image, or a fuzzy texture.

When all of these neurons pass their estimates onto the next hidden layer, those neurons may be trained to look for more complex components. Like, one hidden neuron may check whether there’s a shape that might be a dog nose. It probably doesn’t care about data from previous layers that looked for furry textures, so it weights those by 0 or close to 0. But it may really care about neurons that looked for the “top of the nose” and “bottom of the nose” and “nostrils”. It weights those by large positive numbers. Again, it would add up all the weighted values from the previous layer neurons, squish the value to be between 0 and 1, and pass this to the next layer. That’s the gist of the math, but we’re simplifying a bit.



It’s important to know that neural networks don’t actually understand ideas like “nose” or “eyelid.” Each neuron is doing a calculation on the data it’s given and just flagging specific patterns of light and dark. After a few more hidden layers, we reach the output layer with one neuron! So, after one more weighted addition of the previous layer’s data, which happens in the output neuron, the network should have a good estimate if this image is a dog.



Which means, *Airor* should have a decision.

Airor: Output neuron value: 0.93.

Probability that this is a dog: 93%!

Nice job by *Airor!*

Thinking about how a neural network would process just one image makes it clearer why AI needs fast computers. Like I mentioned before, each pixel in a color image will be represented by 3 numbers - how much red, green, and blue it has. To process a 1000 by 1000 pixel image, which in comparison is a small 3 by 3 inch photo, a neural network needs to look at 3 million features!

AlexNet needed more than 60 million neurons to achieve this, which is a ton of math and could take a lot of time to compute. Which is something we should keep in mind when designing neural networks to solve problems. People are really excited about using deeper neural networks, which are networks with more hidden layers, to do **deep learning** .

Deep networks can combine input data in more complex ways to look for more complex components, and solve trickier problems.

But we can't make all networks like a billion layers deep, because more hidden layers means more math which again would mean that we need faster computers. Plus, as a network get deeper, it gets harder for us to make sense of why it's giving the answers it does. Each neuron in the first hidden layer is looking for some specific component of the input data. But in deeper layers, those components get more abstract from how humans would describe the same data. Now, this may not seem like a big deal, but if a neural network was used to deny our loan request for example, we'd want to know why. Which features made the difference? How were they weighed towards the final answer?

In many countries, we have the legal right to understand why these kinds of decisions were made. And neural networks are being used to make more and more decisions about our lives. Most banks for example use neural networks to detect and prevent fraud. Many cancer tests, like the Pap test for cervical cancer, use a neural network to look at an image of cells under a microscope, and decide whether there's a risk of cancer. And neural networks are how Alexa understands what song you're asking her to play and how Facebook suggests tags for our photos. Understanding how all this happens is really important to being a human in the world right now, whether or not you want to build your own neural network.

This was a lot of big-picture stuff, but the program we gave *Airor* had already been trained to recognize dogs. The neurons already had algorithms that weighted inputs. Next, we'll talk about the learning process used by neural networks to get to the right weights for every neuron, and why they need so much data to work well.

Training Neural Networks

One way to make an artificial brain is by creating a neural network, which can have millions of neurons and billions (or trillions) of connections between them. Nowadays, some neural networks are fast and big enough to do some tasks even better than humans can, like for example playing chess or predicting the weather! But as we know, neural networks don't just work on their own. They need to learn to solve problems by making mistakes. Sounds kind of like us, right?

Neural networks handle mistakes by using an algorithm called **backpropagation** to make sure all the neurons that contributed to an error get their math adjusted, neural networks have two main parts:

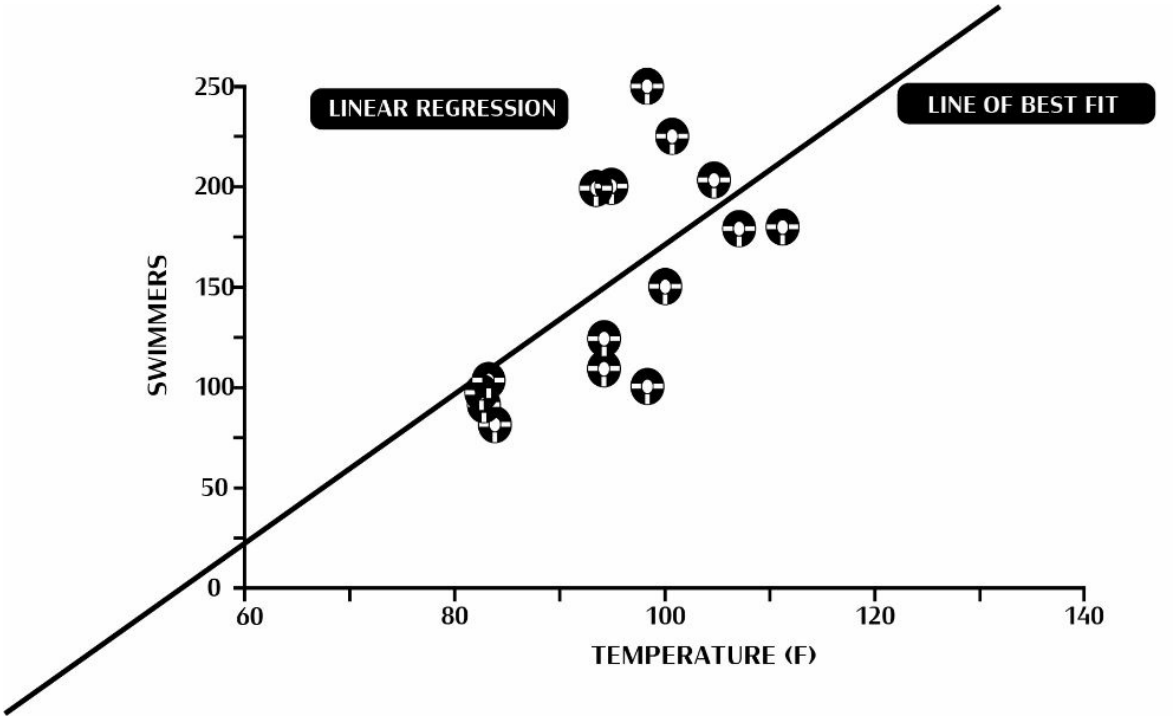
1. The architecture and
2. The weights

The architecture includes neurons and their connections. And the weights are numbers that fine-tune how the neurons do their math to get an output. So, if a neural network makes a mistake, this often means that the weights aren't adjusted correctly and we need to update them so they make better predictions next time.

The task of finding the best weights for a neural network architecture is called **optimization** . And the best way to understand some basic principles of optimization is with an example with the help of my pal *Airor* .

Say that I manage a swimming pool, and I want to predict how many people will come next week, so that I can schedule enough lifeguards. A simple way to do this is by graphing some data points, like the number of swimmers and the temperature in celsius for every day over the past few weeks. Then, we can look for a pattern in that graph to make predictions. The way computers do this is with an optimization strategy called linear regression. We start by drawing a random straight line on the graph, which

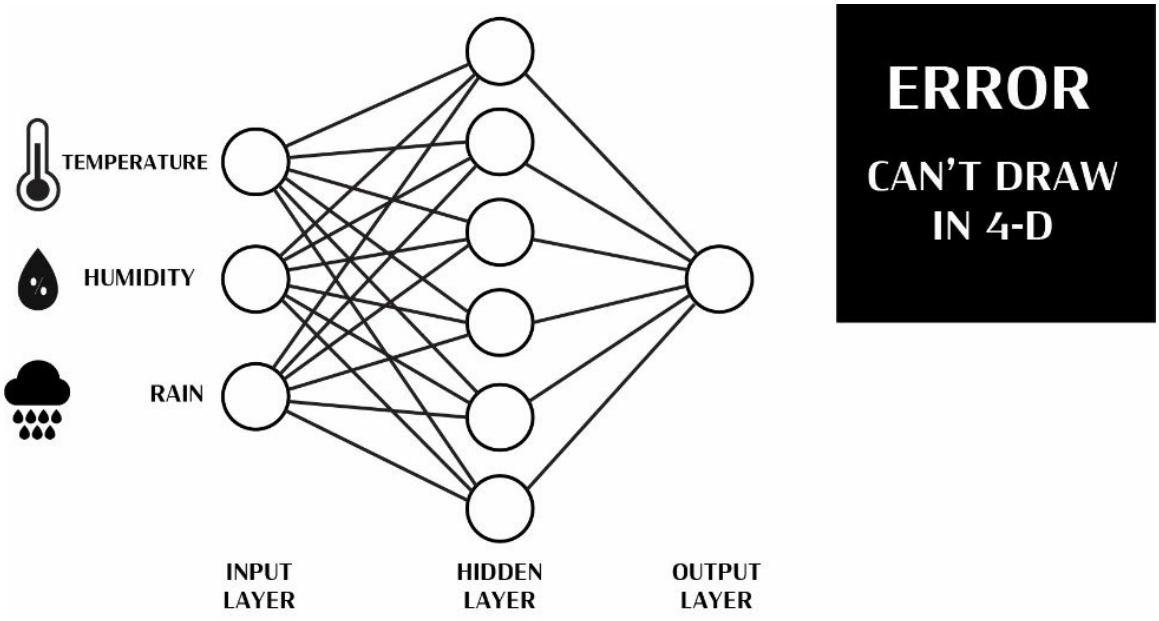
kind of fits the data points. To optimize though, we need to know how incorrect this guess is. So, we calculate the distance between the line and each of the data points, add it all up, and that gives us the error. We're quantifying how big of a mistake we made. The goal of linear regression is to adjust the line to make the error as small as possible. We want the line to fit the training data as much as it can. The result is called the line of best fit.



We can use this straight line to predict how many swimmers will show up for any temperature, but parts of it defy logic. For example, super cold days have a negative number, while dangerously hot days have way more people than the pool can handle. To get more accurate results, we might want to consider more than two features, like for example adding the humidity which

would turn our 2d graph into 3d. And our line of best fit would be more like a plane of best fit. But if we added a fourth feature, like whether it's raining or not, suddenly we can't visualize this anymore. So, as we consider more features, we add more dimensions to the graph, the optimization problem gets trickier, and fitting the training data is tougher. This is where neural networks come in handy. Basically, by connecting together many simple neurons with weights, a neural network can learn to solve complicated problems, where the line of best fit becomes a weird multi-dimensional function.

Let's give *Airor* an untrained neural network. To stick with the same example, the input layer of this neural network takes features like temperature, humidity, rain, and so on. And the output layer predicts the number of swimmers that will come to the pool. We're not going to worry about designing the architecture of Airor's neural network right now. Let's just focus on the weights. He'll start, as always, by setting the weights to random numbers, like the random line on the graph we drew earlier. Only this time, it's not just one random line. Because we have lots of inputs, it's lots of lines that are combined to make one big, messy function. Overall, this neural network's function resembles some weird multi-dimensional shape that we don't really have a name for.



To train this neural network, we'll start by giving *Airor* a bunch of

measurements from the past 10 days at the swimming pool, because these are the days where we also know the output attendance. We'll start with one day, where it was 80 degrees Fahrenheit, 65% humidity, and not raining (which we'll represent with 0). The neurons will do their thing by multiplying those features by the weights, adding the results together, and passing information to the hidden layers until the output neuron has an answer. What do you think, *Airor?*

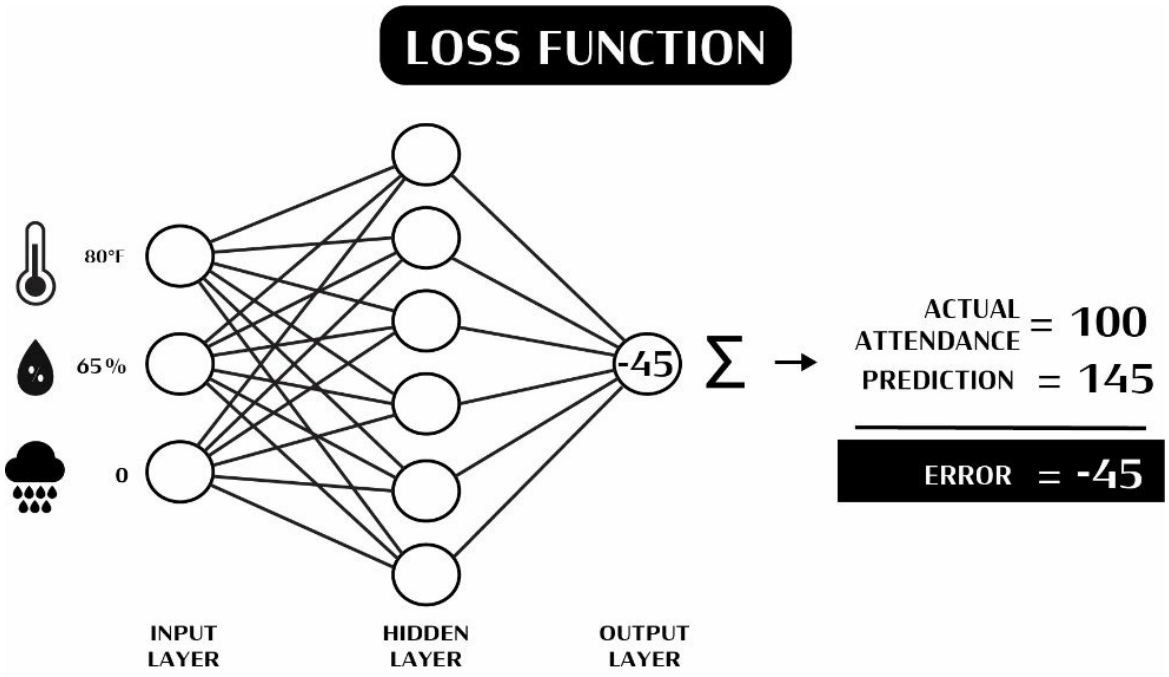
Airor: 145 people were at the pool!

Just like before, there is a difference between the neural network's output and the actual swimming pool attendance which was recorded as 100 people. Because we just have one output neuron, that difference of 45 people is the error.

$$\text{Error} = 145 - 100 = 45$$

Pretty simple. In some neural networks though, the output layer may have a lot of neurons. So, the difference between the predicted answer and the correct answer is more than just one number. In these cases, the error is represented by what's known as a loss function. Moving forward, we need to adjust the neural network's weights so that the next time we give *Airor* similar inputs, his math and final output will be more accurate.

Basically, we need *Airor* to learn from his mistakes, a lot like when we pushed a button to supervise his learning when he had the perceptron program. But this is trickier because of how complicated neural networks are. To help neural networks learn, scientists and mathematicians came up with an algorithm called backpropagation of the error, or just **backpropagation** . The basic goal is to look at the loss function and then assign blame to neurons back in the previous layers of the network. Some neurons' calculations may have been more to blame for the error than others, so their weights will be adjusted more. This information is fed backwards, which is where the idea of backpropagation comes from. So, for example, the error from our output neuron would go back a layer and adjust the weights that get applied to our hidden layer neuron outputs. And the error from our hidden layer neurons would go back a layer and adjust the weights that get applied to our features.



Remember: our goal is to find the best combination of weights to get the lowest error. To explain the logic behind optimization with a metaphor, let's send *Airor* on a metaphorical journey.

Let's imagine that weights in our neural network are like latitude and longitude coordinates on a map. And the error of our neural network is the altitude lower is better. *Airor* the explorer is on a quest to find the lowest poin in the deepest valley. The latitude and longitude of that lowest point where the error is the smallest -- are the weights of the neural network's global optimal solution. But *Airor* has no idea where this valley actually is. By randomly setting the initial weights of our neural network, we're basically dumping him in the middle of the jungle. All he knows is his current latitude, longitude, and altitude. Maybe we got lucky and he's on the side of the deepest valley. But he could also be at the top of the highest mountain far away. The only way to know is to explore! Because the jungle is so dense, it's hard to see very far. The best *Airor* can do is look around and make a guess. He notices that he can descend down a little by moving northeast, so he takes a step down and updates his latitude and longitude. From this new position, he looks around and picks another step that decreases his altitude a little more. And then another… and another. With every brave step, he

updates his coordinates and decreases his altitude.

Eventually, *Airor* looks around and finds that he can't go down anymore. He celebrates, because it seems like he found the lowest point in the deepest valley! Or... so he thinks. *Airor* only found the bottom of a small gorge when he ran out of "down." It's way better than where he started, but it's definitely not the lowest point of the deepest valley. So, he just found a local optimal solution, where the weights make the error relatively small, but not the smallest it could be.

Backpropagation and learning always involves lots of little steps, and optimization is tricky with any neural network. If we go back to our example of optimization as exploring a metaphorical map, we're never quite sure if we're headed in the right direction or if we've reached the lowest valley with the smallest error again that's the global optimal solution. But tricks have been discovered to help us better navigate.

For example, when we drop an explorer somewhere on the map, they could be really far from the lowest valley, with a giant mountain range in the way. So, it might be a good idea to try different random starting points to be sure that the neural network isn't getting stuck at a locally optimal solution. Or instead of restarting over and over again, we could have a team of explorers that start from different locations and explore the jungle simultaneously. This strategy of exploring different solutions at the same time on the same neural network is especially useful when you have a giant computer with lots of processors. And we could even adjust the explorer's step size, so that they can step right over small hills as they try to find and descend into a valley. This step size is called the **learning rate** , and it's how much the neuron weights get adjusted every time backpropagation happens.

We're always looking for more creative ways to explore solutions, try different combinations of weights, and minimize the loss function as we train neural networks. But even if we use a bunch of training data and backpropagation to find the global optimal solution… we're still only halfway done. The other half of training an AI is checking whether the system can answer new questions. It's easy to solve a problem we've seen before, like taking a test after studying the answer key. We may get an A, but we didn't actually learn much. To really test what we've learned, we need to solve problems we haven't seen before. Same goes for neural networks.

This whole time, *Airor* has been training his neural network with swimming pool data. His neural network has dozens of features like temperature, humidity, rain, day of the week, and wind speed… but also grass length, number of butterflies around the pool, and the average GPA of the lifeguards. More data can be better for finding patterns and accuracy, as long as the computer can handle it! Over time, backpropagation will adjust the neuron weights, so that neural network's output matches the training data. Remember, that's called fitting to the training data, and with this complicated neural network, we're looking for a multi-dimensional function. And sometimes, backpropagation is too good at making a neural network fit to certain data.

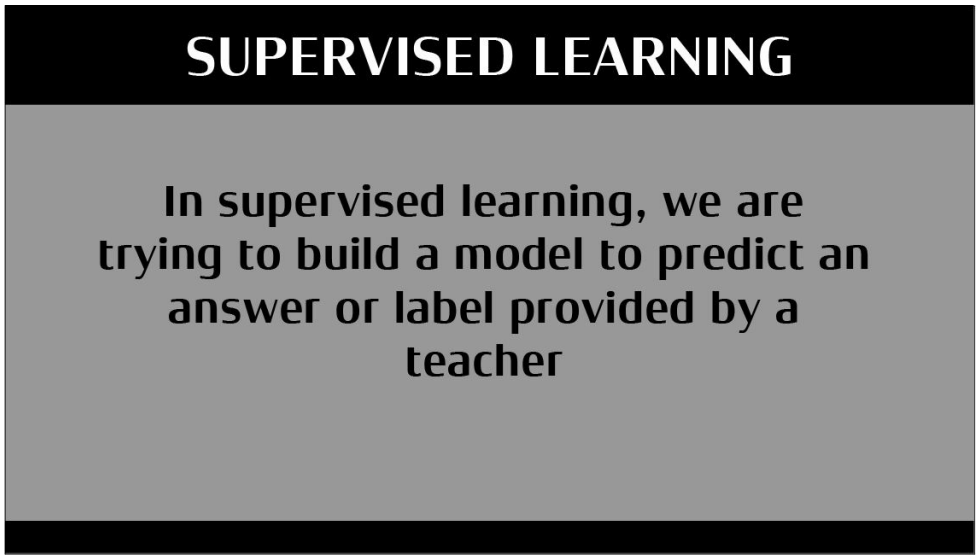
Neural networks are really good at finding these kinds of relationships. And it can be a big problem, because if we give a neural network some new data that doesn't adhere to these silly correlations, then it will probably make some strange errors. That's a danger known as **overfitting** . The easiest way to prevent overfitting is to keep the neural network simple. If we retrain *Airor's* swimming pool program without data like grass length and number of butterflies, and we observe that our accuracy doesn't change, then ignoring those features is best. So, training a neural network isn't just a bunch of math! We need to consider how to best represent our various problems as features in AI systems, and to think carefully about what mistakes these programs might make. Next time, we'll jump into our very first lab, where we'll apply all this knowledge and build a neural network together.

Unsupervised Learning

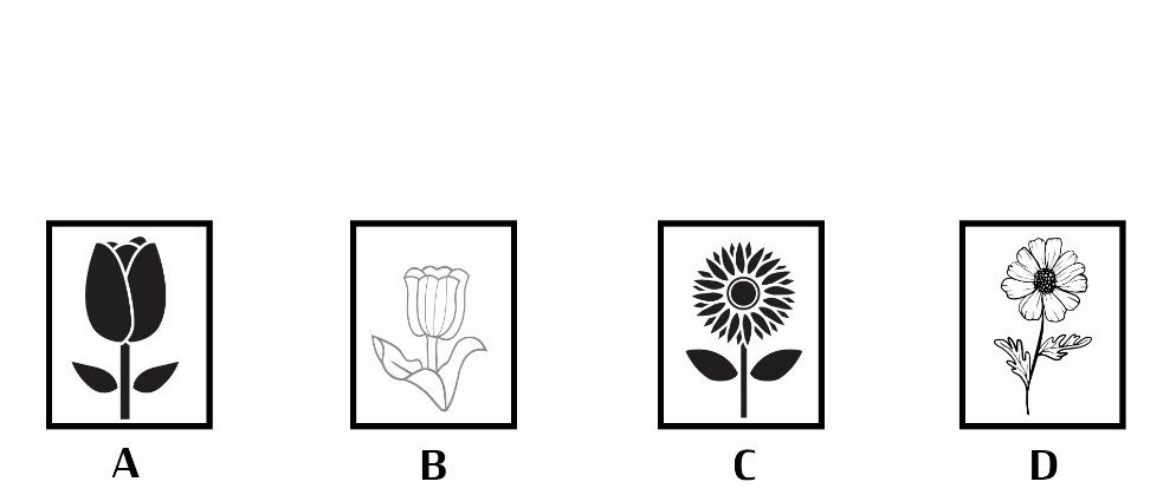
So far, we've focused on artificial intelligence that uses Supervised Learning. These programs need a teacher to use labelled data to tell them "right" from "wrong." And we humans have places where supervised learning happens, like classrooms with teachers, but that's not the only way we learn. We can also learn lots of things on our own by finding patterns in the world.

We can look at dogs and elephants and know they're different animals without anyone telling us. Or we can even figure out the rules of a sport just by watching people play. This kind of learning without a teacher is called **Unsupervised Learning** and, in some cases, computers can do it too.

The key difference between supervised and unsupervised learning is what we're trying to predict.


SUPERVISED LEARNING
In supervised learning, we are trying to build a model to predict an answer or label provided by a teacher

In unsupervised learning, instead of a teacher, the world around us is basically providing training labels. Unsupervised learning is about modelling the world by guessing like this, and it’s useful because we don’t need labels provided by a teacher. Babies do a lot of unsupervised learning by watching and imitating people, and we’d like computers to be able to learn like this as well. This lets us utilize lots of freely available data in the world or on the internet. In many cases, one of the easiest ways to understand how AI can use unsupervised learning is by doing it ourselves, so let’s look at a few photos of flowers with no labels.



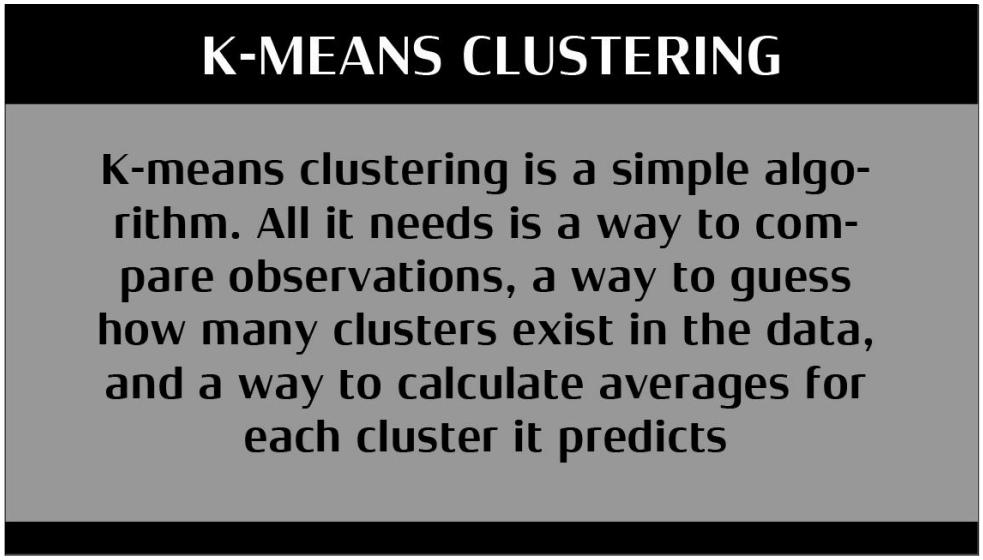
The most basic way to model the world is to assume that it's made up of distinct groups of objects that share properties. So, for example, how many types of flowers are here? We could say there are two because there are two colors, purple (A , D) and yellow (B, C). Or we could look at the petal shapes, and divide them into round petals and tall vertical ones. Or maybe we have some more experience with flowers and realize that two of these are tulips, one is a sunflower, and one is a daisy, so there are three categories.

Immediately recognizing different properties like this and creating categories is called unsupervised clustering. We don't have labels provided by a teacher, but we do have a key assumption about the world that we're modelling: certain objects are more similar to each other than others. We can program computers to perform clustering too. But to do that, we need to choose a few properties of flowers we're interested in looking at, like how we picked color or shape just now.

For a more realistic example, let's say I bought a packet of iris seeds to plant in my garden. After the flowers bloom though, it looks like there were several species of irises mixed up in that one packet. Now I'm no expert gardener, but I can use some AI to help me analyse my garden. To construct a model, we have to answer two key questions. First, what observations can we measure? All of these flowers are purple, so that's probably not the best way

to tell them apart. But different irises seem to have different petal lengths and widths, which we can measure and place on the graph with petal length on the Y axis and width on the X axis. And second, how do we want to represent the world? We're going to stick to a very simple assumption here: there are clusters in our data.

Specifically, we're going to say there are some number of groups called K clusters, but we don't know where they are. To help us, we're going to use the K-means clustering algorithm.


K-MEANS CLUSTERING
K-means clustering is a simple algorithm. All it needs is a way to compare observations, a way to guess how many clusters exist in the data, and a way to calculate averages for each cluster it predicts

In particular, we want to calculate the mean by adding up all data points in a cluster and dividing by the total number of points. Remember, unsupervised learning is about modelling the world, so our algorithm will have two steps:

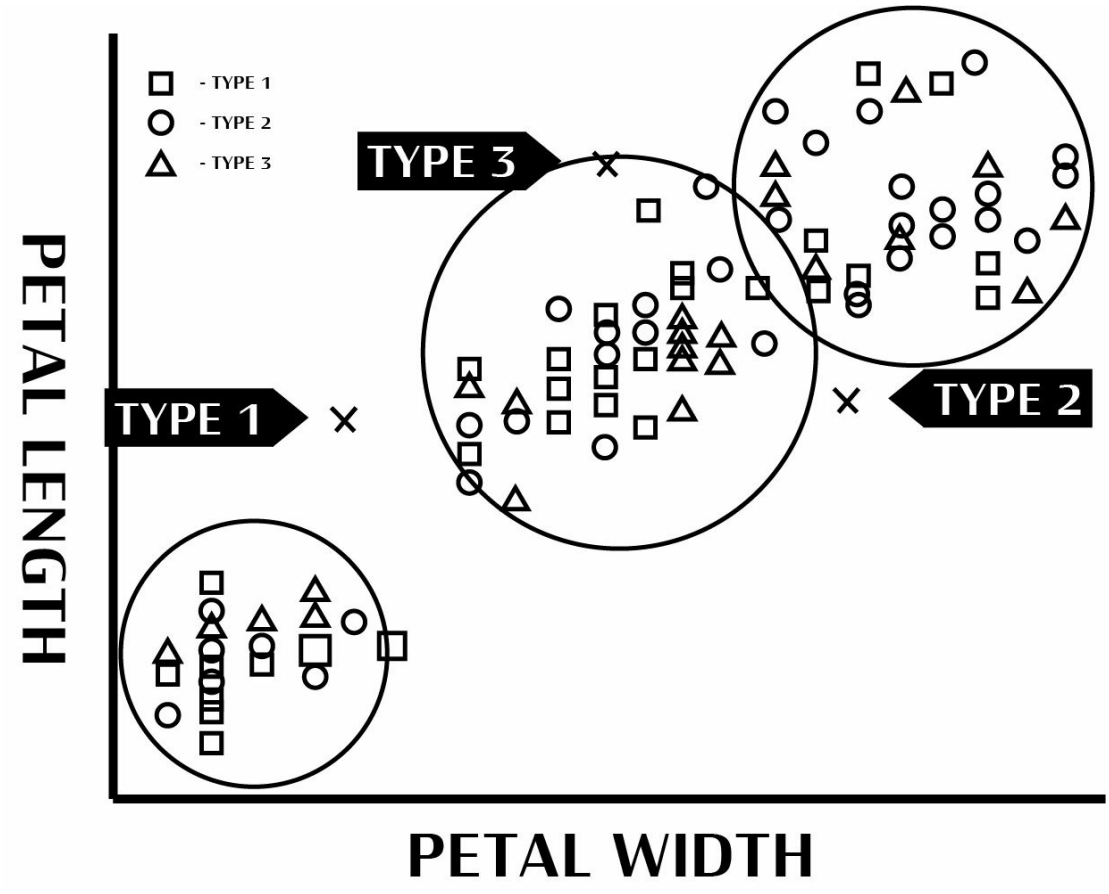
First, our AI will predict. What does the model expect the world to look

like? In other words, which flowers should be clustered together because they're the same species?

Second, our AI will correct or learn. The model will update its beliefs to agree with its observation of the world.

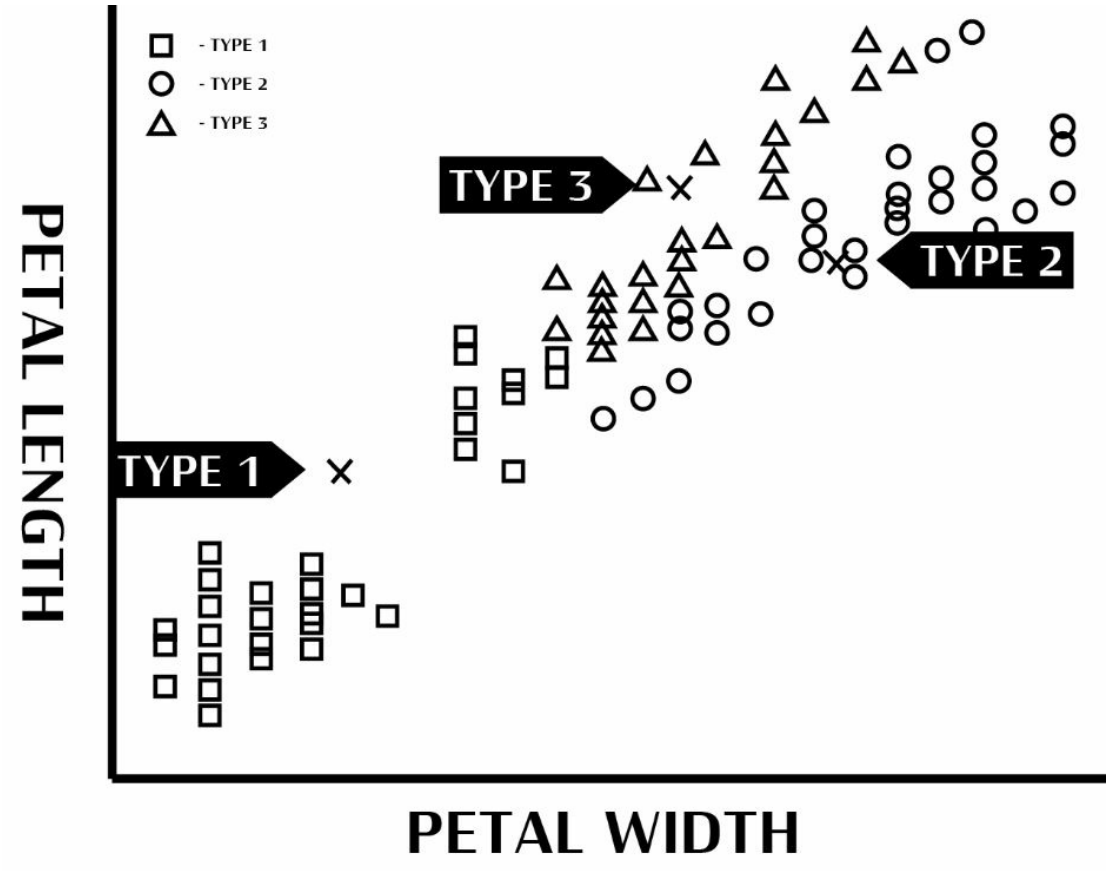
To start the process, we have to specify how many clusters the model should look for. I'm guessing there are three clusters in the data, so that becomes the model's initial understanding of the world, and we're looking for K=3 averages, or three types of irises. But to start, our model doesn't really know anything, so the averages are random and so are its predictions. Each datapoint (which is a flower) is given a label as type1, type2, or type3, based on the algorithm's beliefs.

Next, our model tries to correct itself. The average of each cluster of datapoints should be in the middle, so the model corrects itself by calculating new averages. We can see those averages in the below graph, marked with Xs, which gives our updated model of the three (or so we guessed) types of irises.



The graph is still pretty noisy. For example, it's a little weird that there are type2 flowers so close to the average for type3. But we did start with a random model, so we can't expect too much accuracy. Logically, we know that irises of the same species tend to have similar petals, so those datapoints should be clustered together. Since we just did a correction or learning step, we can repeat the process, starting with a new prediction step.

Let's predict new labels using the Xs that mark the averages of each label. We'll give every datapoint the label of its closest X type1, type2, or type3 -- and then we'll calculate new averages.



That’s better, but still not the cleanest clusters, so we can repeat the process again: Predict, Learn, Predict, Learn. Eventually, the Xs will stop moving and we have a model of iris clusters created with unsupervised learning!

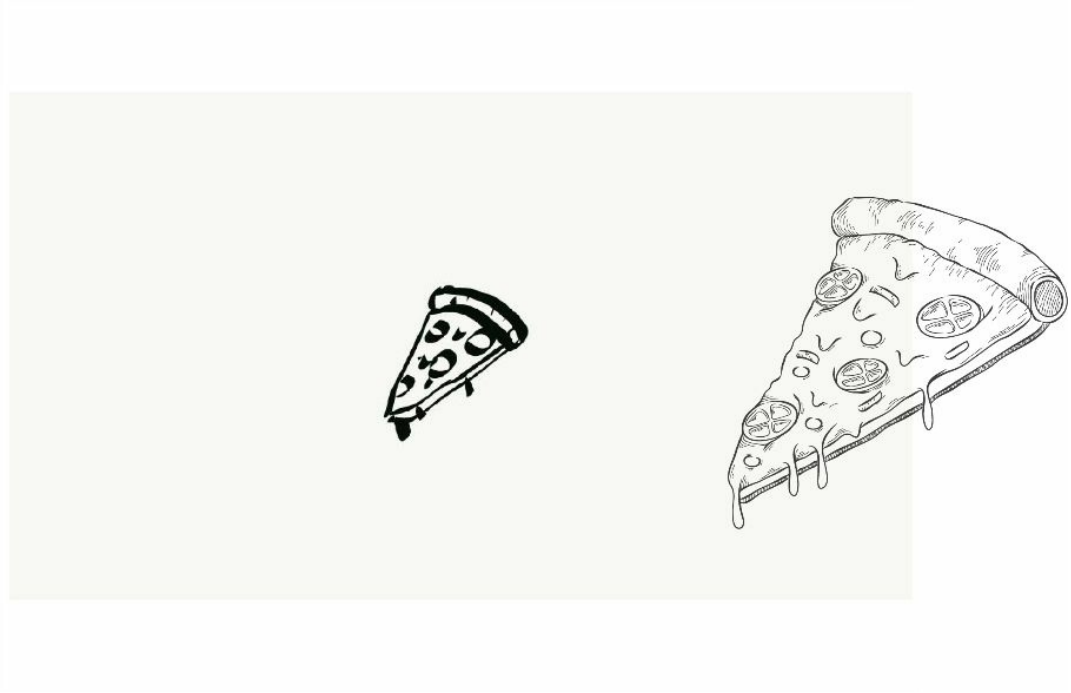
Now the ultimate question is, did we find meaningful patterns about the world with our AI? We made an assumption that there were three types of irises, and we assumed that they have different petal lengths and widths. Was this true? Lucky for us, I have a friend who is a master gardener. I showed him the real-life flowers closest to each of the three averages and he said that type1 is Versicolor, type2 is Setosa and type3 is Virginica. Three different iris species! We learned about the world from observation, which is what makes this unsupervised learning, even though we relied a tiny bit on a teacher(the master gardener) for confirmation and help.

Now that we’ve learned the basics, we can experiment with harder examples. Let’s say we want to use an unsupervised learning algorithm to sort a bunch of different photos, not just three iris species. First, what observations can we measure? How much green there is? Whether there’s a

nose and fur? To have a computer make these observations, we need to measure thousands of red, green, and blue pixels in each image. Second, how do we want to represent the world? Before, we were only working with 2 features, so we could just use averages of the clustered datapoints and get meaningful abstraction from it. But when dealing with images, we can't use the same method, because we won't get much meaning out of averaging colored pixels for what we want to accomplish.

Somehow, we need the model to create a representation that tells us if two images are similar. There are meaningful patterns in the data that are more abstract than individual pixels, and finding them across many images is called **Representation Learning** . These patterns help us understand what's in the images and how to compare them to each other. Representation learning happens both in supervised and unsupervised learning models, so we can do it with or without labels to find patterns in the world.

To understand the basic idea of representation learning, check out this experiment: I'm gonna look at a picture really fast and then try to draw it. My eyes took in the picture and remembered important features, so I'm building a representation in my mind. But I can't just show you my thoughts to get feedback on what parts I misremembered, so I have to produce a reconstruction, or draw the original image from memory. Alright, so this is what I've got.



Now let’s compare my drawing to the original image. Let's see triangle slice of pizza, some cheese, some crust. Pretty good. For an AI, making a reconstruction would mean producing all the right pixel values to make a reconstruction. Our K-means clustering algorithm from before, predicted classes for flowers based on how close the datapoints were to the averages. For images, we will have learned image representations instead of averages. After that step, just like before, the AI will have to correct itself. Previously, we updated the K clusters based on how well our predicted labels fit the data. But for images, we’d have to update the model’s internal representations based on its reconstructions.

There are different ways to use unsupervised learning in combination with representation learning so that an AI can compare images. Like, for example, there’s a type of neural network called an **autoencoder** , which uses the same basic principles of weights and biases to process inputs, pass data onto hidden neuron layers, and finally to a prediction output layer. If *Airor* wa programmed with an autoencoder, the input would be an image, the hidden layers would contain representations, and the output would be a full reconstruction of the original image (which gets more accurate the more we train his AI).

Theoretically, I could give *Airor* a representation of a pizza and he could reconstruct the original pizza image. What’s so powerful about unsupervised

learning is that the world is our teacher. By looking around, taking in a lot of data, and predicting what we'll see and hear next, we learn about how the world works and how it should be represented. When asked how AI will fulfill its grand ambitions, 2018 Turing Award Winner Professor Yann LeCun, said: "We all know that unsupervised learning is the ultimate answer."

So, I guess we better keep working on it! Unsupervised learning is a huge area of active research. The human brain is specially designed for this kind of learning and has different parts for vision, language, movement, and so on. These structures and what kinds of patterns our brains look for were developed over billions of years of evolution. But it's really tricky to build an AI that does unsupervised learning, well because AI systems can't learn exactly like human often do, just by watching and imitating.

Someone, like us, has to design the models and tell them how to look for patterns before letting them loose. Next chapter, we'll look at applying similar concepts to AI systems that find patterns in words and language, in what's called Natural Language Processing.

Natural language processing

Language is one of the most impressive things humans do. Languages come in many shapes and sizes, they can be spoken or written, and are made up of different components like sentences, words, and characters that vary across cultures. For instance, English has 26 letters and Chinese has tens-of-thousands of characters. So far, a lot of the problems we've been solving with AI and machine learning technologies have involved processing images, but the most common way that most of us interact with computers is through language.

We type questions into search engines, we talk to our smartphones to set alarms, and sometimes we even get a little help with our Spanish homework from Google Translate. So, we're going to explore the field of Natural Language Processing.

Natural Language Processing, or NLP, mainly explores two big ideas. First, there's Natural Language Understanding, or how we get meaning out of combinations of letters. These are AI that filter your spam emails, figure out if that Amazon search for "apple" was grocery or computer shopping, or instruct your self-driving car how to get to a friend's house. And second, there's Natural Language Generation, or how to generate language from knowledge. These are AI that perform translations, summarize documents, or chat with you. The key to both problems is understanding the meaning of a word, which is tricky because words have no meaning on their own. We assign meaning to symbols.

To make things even harder, in many cases, language can be ambiguous and the meaning of a word depends on the context it's used in. If I tell you to meet me at the bank, without any context, I could mean the river bank or the place where I'm grabbing some cash. If I say "This fridge is great!", that's a totally different meaning from "This fridge was *great*, it lasted a whole week before breaking." So, how did we learn to attach meaning to sounds?

How do we know great [enthusiastic] means something different from great [sarcastic]? Well, even though there's nothing inherent in the word "cat" that tells us it's soft, purrs, and chases mice… when we were kids, someone probably told us "this is a cat." Or a gato, māo, billee, qut. When we're solving a natural language processing problem, whether it's natural language understanding or natural language generation, we have to think about how our AI is going to learn the meaning of words and understand our potential mistakes.

Sometimes we can compare words by looking at the letters they share. This works well if a word has morphology. Take the root word "swim" for example. We can modify it with rules so if someone's doing it right now, they're swimming, or the person doing the action is the swimmer. Drinking, drinker, thinking, thinker, … you get the idea. But we can't use morphology for all words, like how knowing that a van is a vehicle doesn't let us know that a vandal smashed in a car window. Many words that are really similar, like cat and car, are completely unrelated. And on the other hand, cat and Felidae (the word for the scientific family of cats) mean very similar things and only share one letter!

One common way to guess that words have similar meaning is using distributional semantics, or seeing which words appear in the same sentences a lot. This is one of many cases where NLP relies on insights from the field of linguistics. As the linguist John Firth once said, "You shall know a word by the company it keeps." But to make computers understand distributional semantics, we have to express the concept in math.

One simple technique is to use **count vectors** . A count vector is the number of times a word appears in the same article or sentence as other common words. If two words show up in the same sentence, they probably have pretty similar meanings. So, let's say we asked an algorithm to compare three words, car, cat, and Felidae, using count vectors to guess which ones have similar meaning. We could download the beginning of the Wikipedia pages for each word to see which other words show up. Here's what we got:

CAR		CAT		FELIDAE	
170	THE	69	A	120	THE
97	BE	63	CAT	62	AND
93	IN	62	THE	42	BE

84	OF	62	BE	38	OF
81	AND	35	AND	36	A
78	A	30	OF	32	TO
53	TO	28	IN	29	IN
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
17	PATENT	7	PET	5	PREY

A lot of the top words are all the same: the, and, of, in. These are all function words or stop words, which help define the structure of language, and help convey precise meaning. Like how “an apple” means any apple, but “the apple” specifies one in particular. But, because they change the meaning of another word, they don’t have much meaning by themselves, so we’ll remove them for now, and simplify plurals and conjugations. Let’s try it again:

CAR		CAT		FELIDAE	
50	CAR	63	CAT	27	CAT
24	VEHICLE	19	DOMESTIC	17	SPECIES
24	BENZ	7	SPECIES	9	FUR
23	ENGINE	7	PET	8	FELIDAE
17	DAIMLER	7	HUMAN	8	SMALL
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
7	GASOLINE	4	KITTEN	5	PREY
5	RUN	3	FELIDAE	5	CLAW

Based on this, it looks like cat and Felidae mean almost the same thing, because they both show up with lots of the same words in their Wikipedia articles! And neither of them mean the same thing as car. But this is also a really simplified example. One of the problems with count vectors is that we have to store a LOT of data. To compare a bunch of words using counts like this, we’d need a massive list of every word we’ve ever seen in the same sentence, and that’s unmanageable.

So, we’d like to learn a representation for words that captures all the same

relationships and similarities as count vectors but is much more compact. In the unsupervised learning, we talked about how to compare images by building representations of those images. We needed a model that could build internal representations and that could generate predictions. We can do the same thing for words. This is called an **Encoder-Decoder model** : The encoder tells us what we should think and remember about what we just read... and the decoder uses that thought to decide what we want to say or do.

We're going to start with a simple version of this framework. Let's create a little game of fill in the blank to see what basic pieces we need to train an unsupervised learning model. This is a simple task called language modelling.

If I have the sentence:

I'm kinda hungry, I think I'd like some chocolate _____ .

What are the most likely words that can go in that spot? And how might we train a model to encode the sentence and decode a guess for the blank? In this example, I can guess the answer might be "cake" or "milk" but probably not something like "potatoes," because I've never heard of "chocolate potatoes" so they probably don't exist. Definitely don't exist.

The group of words that can fill in that blank is an unsupervised cluster that an AI could use. So, for this sentence, our encoder might only need to focus on the word chocolate so the decoder has a cluster of "chocolate food words" to pull from to fill in the blank.

Now let's try a harder example:

Dianna, a friend of mine from San Diego who really loves physics, is having a birthday party next week, so I want to find a present for ____.

When I read this sentence, my brain identifies and remembers two things: First, that we're talking about Dianna from 27 words ago! And second, that my friend Dianna uses the pronoun "her." That means we want our encoder to build a representation that captures all these pieces of information from the sentence, so the decoder can choose the right word for the blank. And if we keep the sentence going:

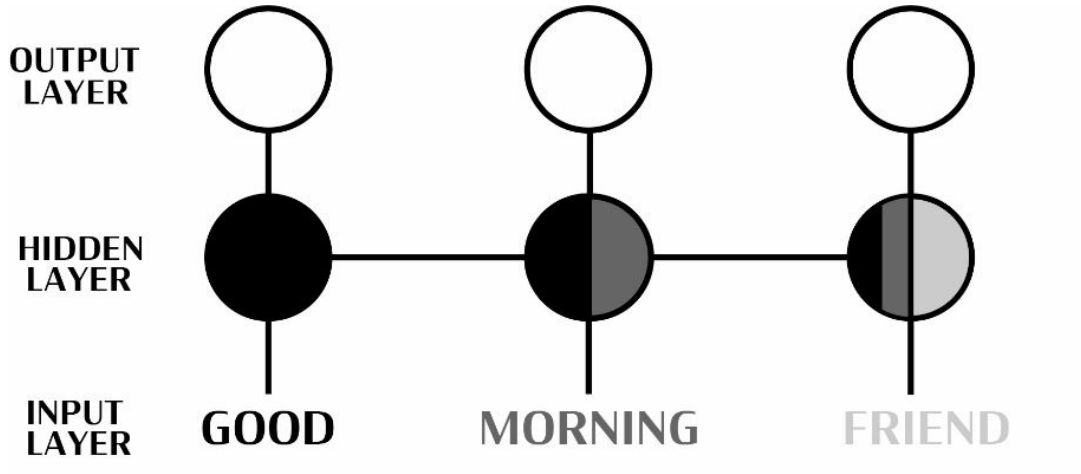
Dianna, a friend of mine from San Diego who really loves physics, is having a birthday party next week, so I want to find a present for her that has to do with _____ .

Now, I can remember that Dianna likes physics from earlier in the sentence. So, we'd like our encoder to remember that too, so that the decoder can use that information to guess the answer. We can see how the representation the model builds really has to remember key details of what we've said or heard. And there's a limit to how much a model can remember.

Professor Ray Mooney has famously said that we'll "never fit the whole meaning of a sentence into a single vector" and we still don't know if we can. Professor Mooney may be right, but that doesn't mean we can't make something useful. So, far we've been using words. But computers don't work words quite like this. So, let's step away from our high level view of language modelling and try to predict the next word in a sentence anyway with a neural network.

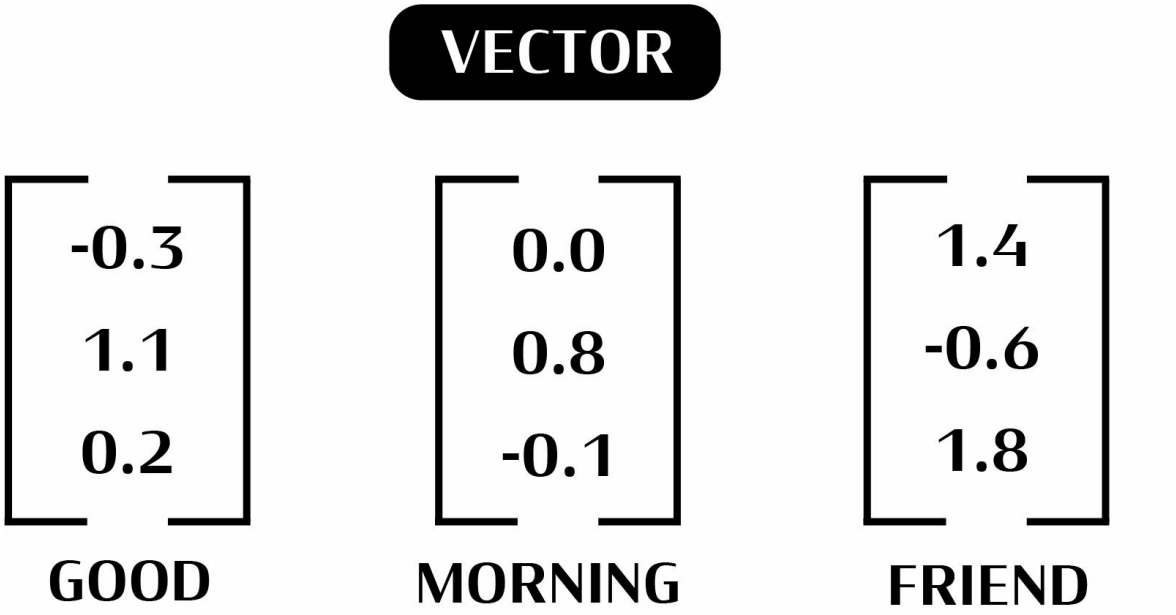
To do this, our data will be lots of sentences we collect from things like someone speaking or text from books. Then, for each word in every sentence, we'll play a game of fill-in-the-blank. We'll train a model to encode up to that blank and then predict the word that should go there. Since we have the whole sentence, we know the correct answer.

First, we need to define the **encoder** . We need a model that can read in th input, which in this case is a sentence. To do this, we'll use a type of neural network called a **Recurrent Neural Network** or **RNN** . RNNs have a loop in them that lets them reuse a single hidden layer, which gets updated as the model reads one word at a time. Slowly, the model builds up an understanding of the whole sentence, including which words came first or last, which words are modifying other words, and a whole bunch of other grammatical properties that are linked to meaning.



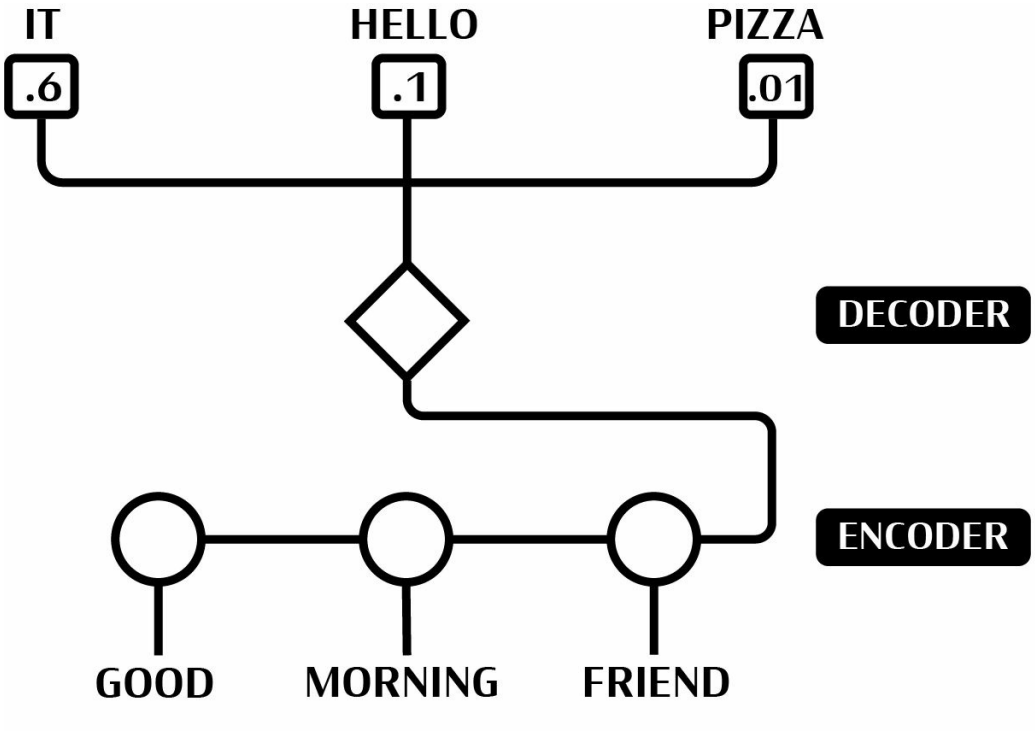
Now, we can't just directly put words inside a network. But we also don't have features we can easily measure and give the model either. Unlike images, we can't even measure pixel values. So, we're going to ask the model to learn the right representation for a word on its own (this is where the unsupervised learning comes in).

To do this, we'll start off by assigning each word a random representation in this case a random list of numbers called a vector.



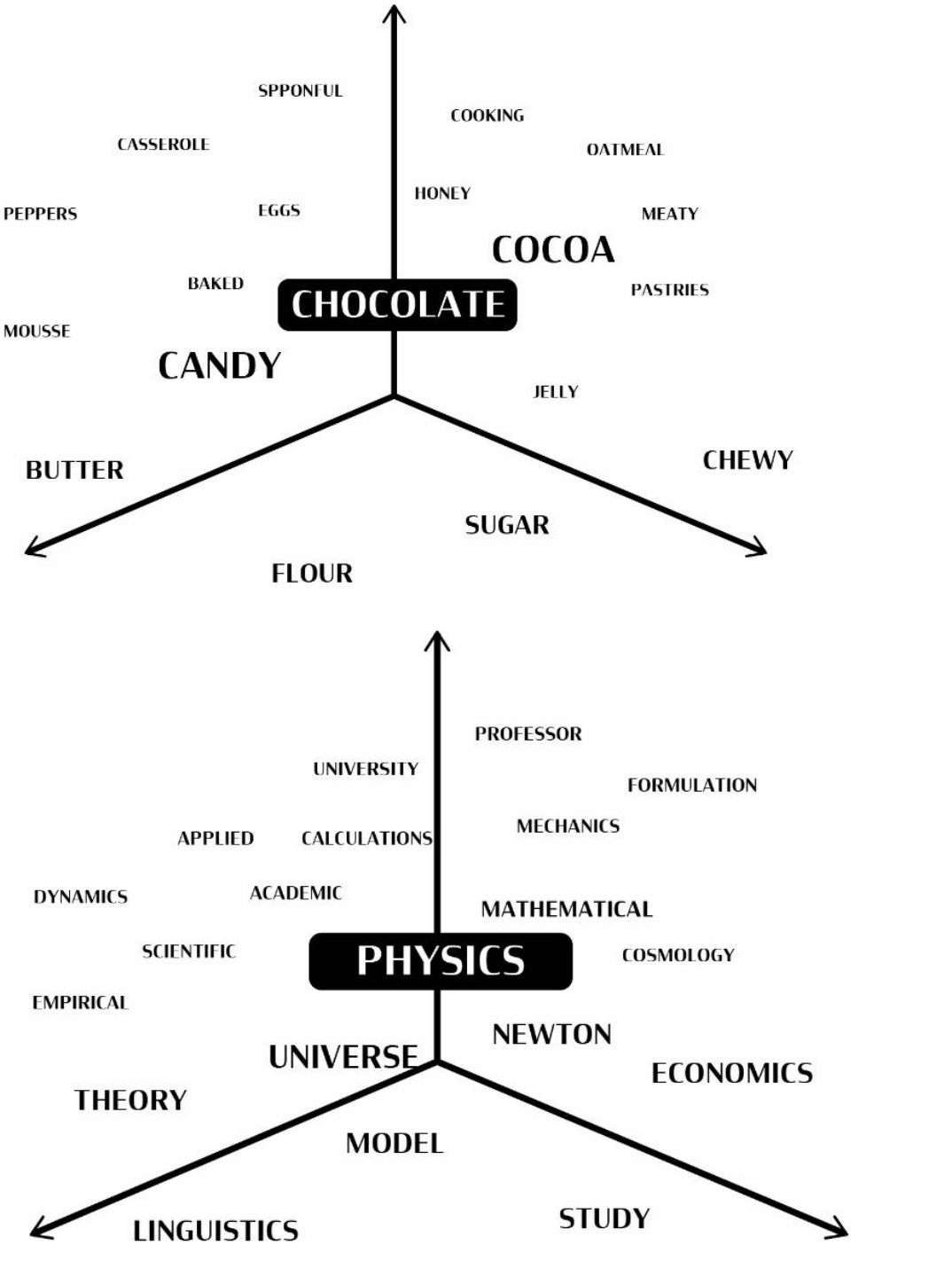
Next, our encoder will take in each of those representations and combine them into a single shared representation for the whole sentence. At this point,

our representation might be gibberish, but in order to train the RNN, we need it to make predictions. For this particular problem, we'll consider a very simple decoder, a single layer network that takes in the sentence representation vector, and then outputs a score for every possible word in our vocabulary. We can then interpret the highest scored word as our model's prediction.



Then, we can use backpropagation to train the RNN, like we've done before with neural networks. So, by training the model on which word to predict next, the model learns weights for the encoder RNN and the decoder prediction layer. Plus, the model changes those random representations we gave every word at the beginning. Specifically, if two words mean something similar, the model makes their vectors more similar. Using the vectors to help make a plot, we can actually visualize word representations.

For example, earlier we talked about chocolate and physics, so let's look at some word representations that researchers at Google trained.



Near “chocolate,” we have lots of foods like cocoa and candy: By comparison, words with similar representations to “physics” are newton and universe. This whole process has used unsupervised learning, and it’s given us a basic way to learn some pretty interesting linguistic representations and

word clusters. But taking in part of a sentence and predicting the next word is just the tip of the iceberg for NLP.

If our model took in English and produced Spanish, we'd have a translation system. Or our model could read questions and produce answers, like Siri or Alexa try to do. Or our model could convert instructions into actions to control a household robot. But the representations of words that our model learns for one kind of task might not work for others. Like, for example, if we trained *Airor* based on reading a bunch of cooking recipes, he might learn that roses are made of icing and placed on cakes. But he won't learn that cake roses are different from real roses that have thorns and make a pretty bouquet.

Acquiring, encoding, and using written or spoken knowledge to help people is a huge and exciting task, because we use language for so many things! Every time you type or talk to a computer, phone or other gadget, NLP is there.

Reinforcement Learning

I want to get a cookie from a jar that's on a tall shelf. There isn't one "right way" to get the cookies. Maybe I find a ladder, use a lasso, or build a complicated system of pulleys. These could all be brilliant or terrible ideas, but if something works, I get the sweet taste of victory... and I learn that doing that same thing could get me another cookie in the future. We learn lots of things by trial-and-error, and this kind of "learning by doing" to achieve complicated goals is called **Reinforcement Learning** .

So far, we've talked about two types of learning: **Supervised Learning** , where a teacher gives an AI answers to learn from, and **Unsupervised Learning** , where an AI tries to find patterns in the world. Reinforcement Learning is particularly useful for situations where we want to train AI's to have certain skills we don't fully understand ourselves. For example, I'm pretty good at walking, but trying to explain the process of walking is kind of difficult. What angle should your femur be relative to your foot? And should you move it with an average angular velocity of… yeah, never mind… its really difficult.

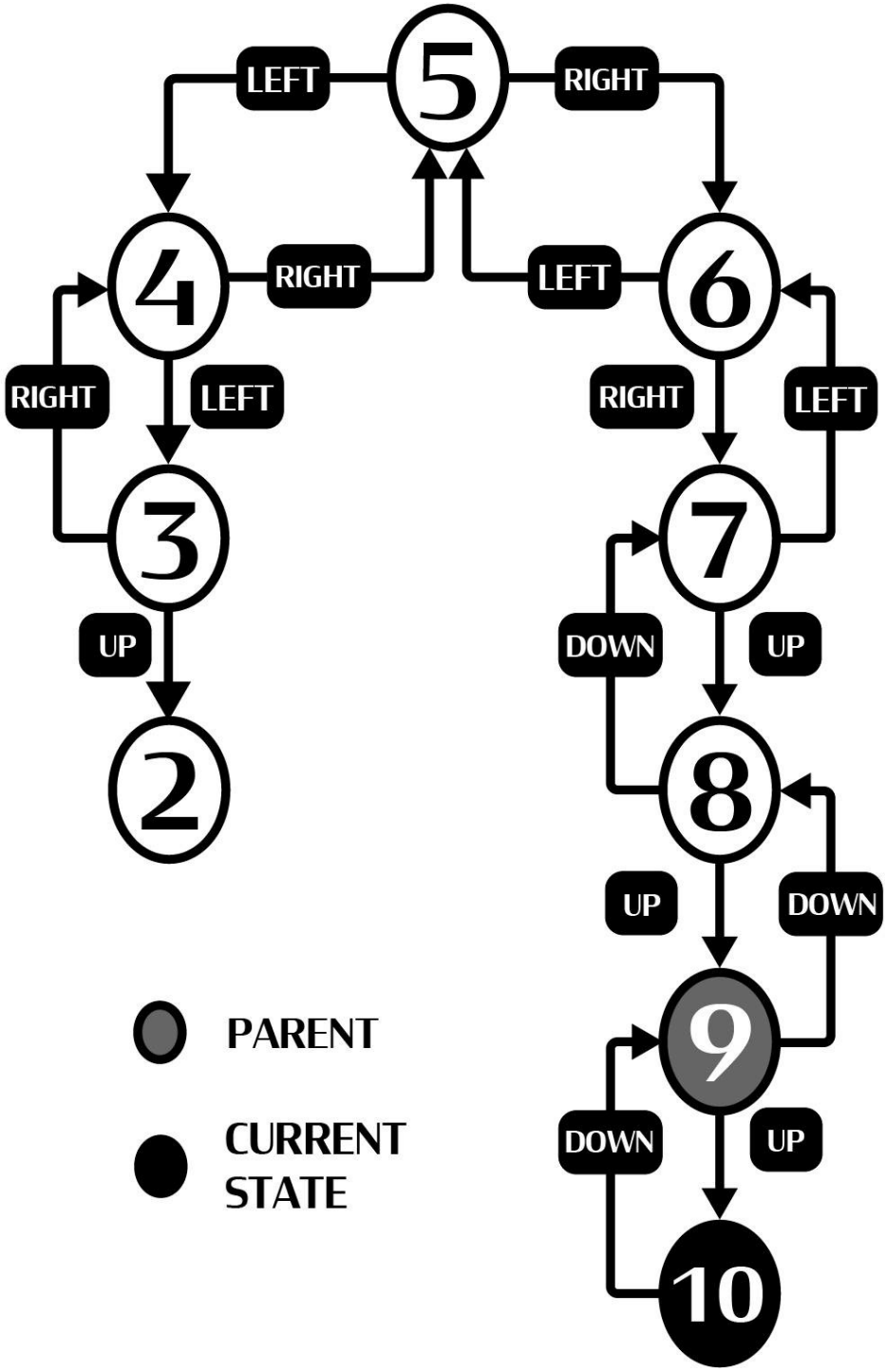
With reinforcement learning, we can train AI's to perform complicated tasks. But unlike other techniques, we only have to tell them at the very end of the task if they succeeded, and then ask them to tell us how they did it. (We're going to focus on this general case, but sometimes this feedback could come earlier.) So, if we want an AI to learn to walk, we give them a reward if they're both standing up and moving forward, and then figure out what steps they took to get to that point. The longer the AI stands up and moves forward, the longer it's walking, and the more reward it gets. So, you can kind of see how the key to reinforcement learning is just trial-and-error, again and again. For humans, a reward might be a cookie or the joy of winning a board game. But for an AI system, a reward is just a small positive signal that basically tells it "good job" and "do that again"! **Google**

Deepmind got some pretty impressive results when they used reinforcement learning to teach virtual AI systems to walk, jump, and even duck under obstacles. It looks kinda silly, but works pretty well! Other researchers have even helped real life robots learn to walk. Seeing the end result is pretty fun and can help us understand the goals of reinforcement learning. But to really understand how reinforcement learning works, we have to learn new language to talk about these AI and what they're doing.

Let's think we have an AI (or Agent) as our loyal subject that's going to learn. An agent makes predictions or performs Actions, like moving a tiny bit forward, or picking the next best move in a game. And it performs actions based on its current inputs, which we call the State. In supervised learning, after each action, we would have a training label that tells our AI whether it did the right thing or not. We can't do that here with reinforcement learning, because we don't know what the "right thing" actually is until it's completely done with the task. This difference actually highlights one of the hardest parts of reinforcement learning called **credit assignment** . It's hard to know which actions helped us get to the reward (and should get credit) and which actions slowed down our AI when we don't pause to think after every action.

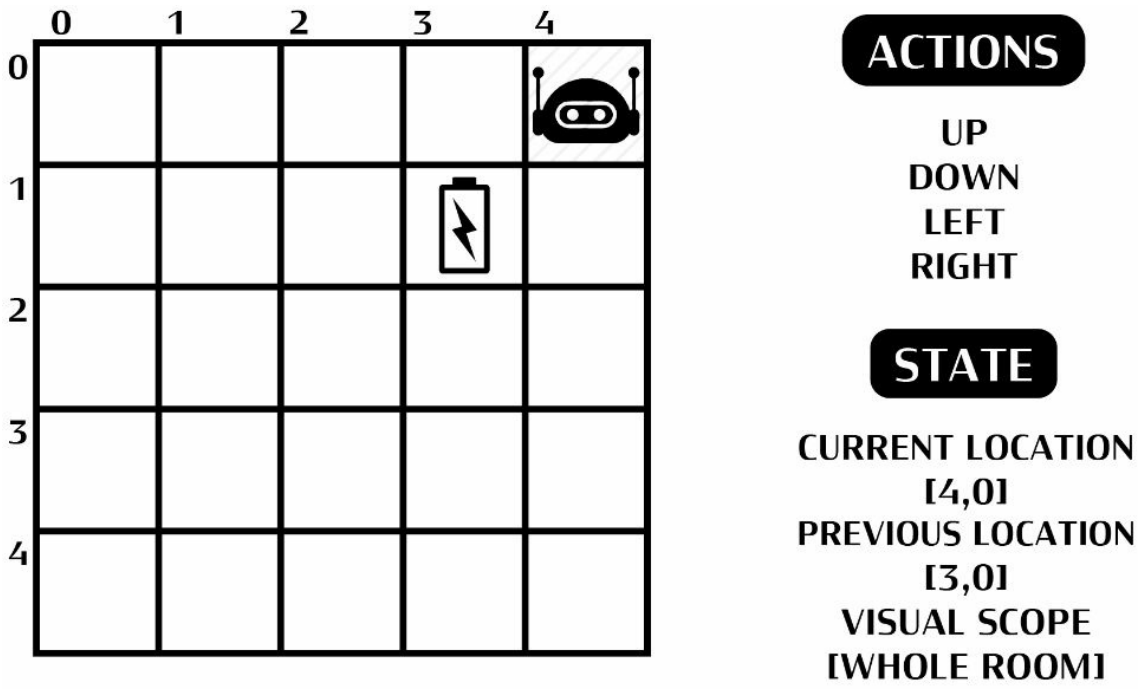
So, the agent ends up interacting with its Environment for a while, whether that's a game board, a virtual maze, or real life kitchen. And the agent takes many actions until it gets a Reward, which we give out when it wins a game or gets that cookie jar from that really tall shelf. Then, every time the agent wins (or succeeds at its task), we can look back on the actions it took and slowly figure out which game states were helpful and which weren't. During this reflection, we're assigning Value to those different game states and deciding on a Policy for which actions work best. We need Values and Policies to get anything done in reinforcement learning.

STATES

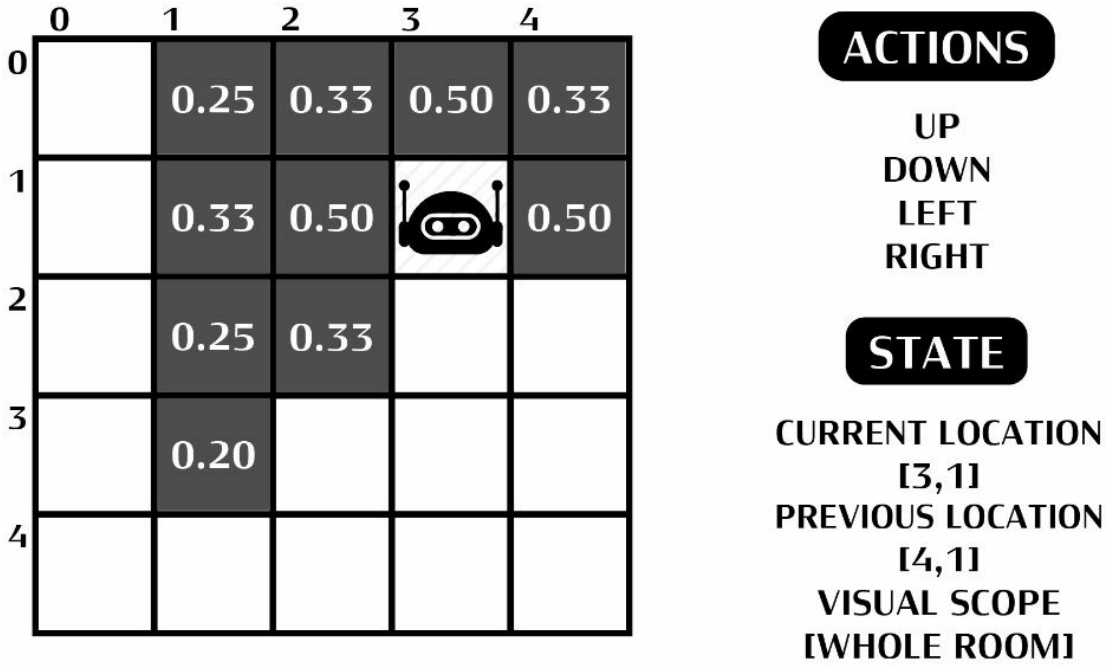

5
LEFT
RIGHT
4
RIGHT
LEFT
6
RIGHT
LEFT
RIGHT
LEFT
3
7
UP
DOWN
UP
2
8
UP
DOWN
PARENT
9
CURRENT STATE
DOWN
UP
10

Let’s say I see some food in the kitchen: a box, a small bag, and a plate with a donut. So, my brain can assign each of these a value, a numerical yummy-ness value. The box probably has 6 donuts in it, the bag probably has 2, and the plate just has 1… so the values I assign are 6, 2, and 1. Now that I’ve assigned each of them a value, I can decide on a policy to plan what action to take! The simplest policy is to go to the highest value (that box of possibly 6 donuts). But I can’t see inside of it, and that could be a box of bagels, so it’s high reward but high risk. Another policy could be low reward but low risk, going with the plate with 1 guaranteed delicious donut. Personally, I’d pick a middle-ground policy, and go for the bag because I have a better chance of guessing that there are donuts inside the bag than the box, and a value of 1 donut isn’t enough.

That’s a lot of vocab, so let’s see these concepts in action to help us remember everything. Our example is going to focus on a mathematical framework that could be used with different underlying machine learning techniques. Let’s say *Airor* wants to go to the charging station to recharge his batteries. In this example, *Airor* is a brand new Agent, and the room is the Environment he needs to learn about. From where he is now in the room, he has four possible Actions: moving up, down, left, or right. And his State is a couple of different inputs: where he is, where he came from, and what he sees. For this example, we’ll assume *Airor* can see the whole room. So, when he moves up (or any direction), his state changes. But he doesn’t know yet if moving up was a good idea, because he hasn’t reached a goal. So go on, *Airor* ... explore…!

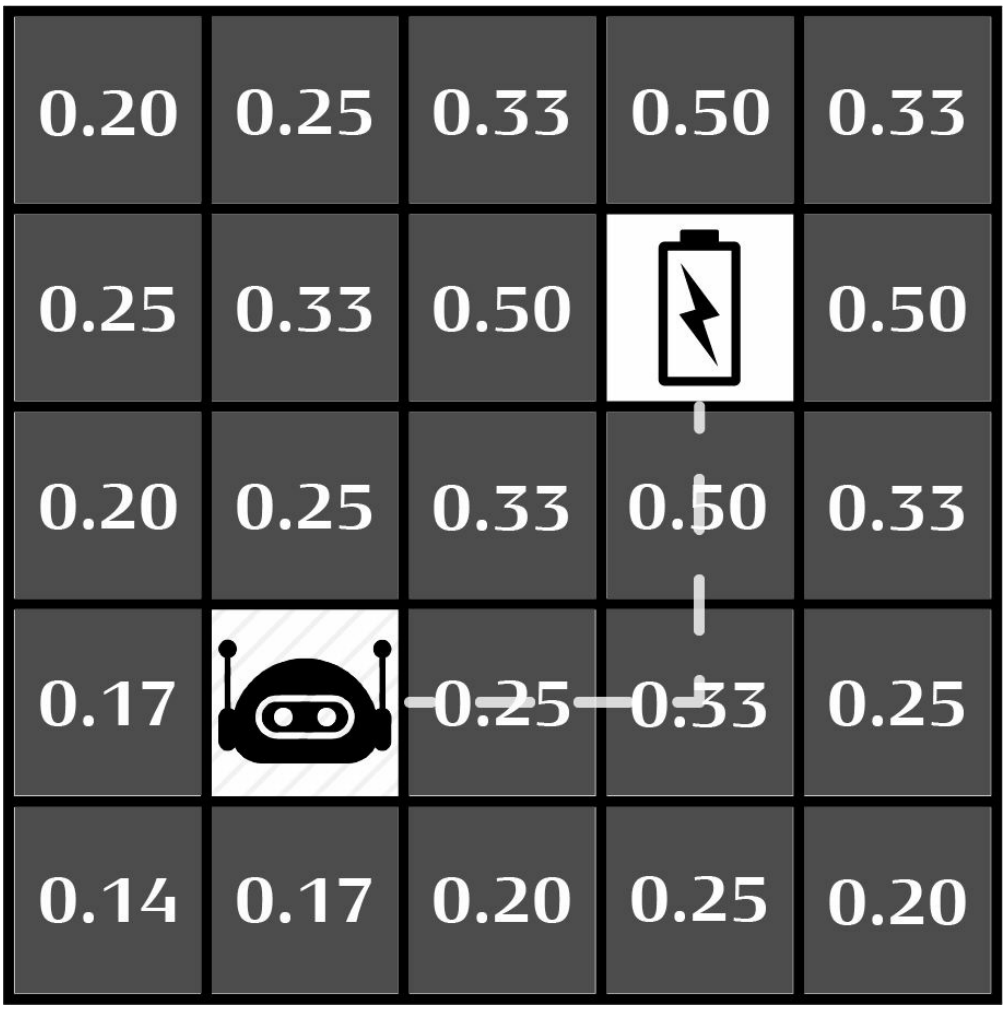


He found the battery, so he got a Reward (that little plus one). Now, we can look back at the path he took and give all the cells he walked through a Value specifically, a higher value for those near the goal, and lower for those farther away.



These higher and lower values help with the trial-and-error of reinforcement learning, and they give our agent more information about better actions to take when he tries again! So, if we put *Airor* back at the start, he'll want to decide on a Policy that maximizes reward. Since he already knows a path to the battery, he'll walk along that path, and he's guaranteed another +1. But that's… too easy. And kind of boring if *Airor* just takes the same long and winding path every time.

So, another important concept in reinforcement learning is the trade-off between exploitation and exploration. Now that *Airor* knows one way to get to the battery, he could just exploit this knowledge by always taking the same 10 actions. It's not a terrible idea he knows he won't get lost and he'll definitely get a reward. But this 10-action path is also pretty inefficient, and there are probably more efficient paths out there. So, exploitation may not be the best strategy. It's usually worth trying lots of different actions to see what happens, which is a strategy called exploration. Every new path *Airor* takes will give him a bit more data about the best way to get a reward. So, let's let *Airor* explore for 100 actions, and after he completes a path, we'll update the values of the cells he's been to. Now we can look at all these new values!



During exploration, *Airor* found a shortcut, so now he knows a path that only takes 4 actions to get to the goal. This means our new policy (which always chooses the best value for the next action) will take *Airor* down this faster path to the target. That's much better than before, but we paid a cost, because during those 100 actions of exploration, he took some paths that were even more inefficient than the first 10-action try and only got a total of 6 points. If *Airor* had just exploited his knowledge of the first path he took for those 100 actions, he could have made it to the battery 10 times and gotten 10 points. So, you could say that exploration was a waste of time.

EXPLORE

ACTIONS	POINTS
18	+1
22	+1
4	+1
14	+1
24	+1
18	+1
100	6

EXPLOIT

ACTIONS	POINTS
10	+1
10	+1
10	+1
10	+1
10	+1
10	+1
10	+1
10	+1
10	+1
10	+1
100	10

But if we started a new competition between the new *Airor* (who knows a 4-action path) and his younger, more foolish self (who knows a 10-action path), over 100 actions, the new *Airor* would be able to get 25 points because his path is much faster. His reinforcement learning helped! So, should we explore more to try and find an even better path? Or should we just use exploitation right away to collect more points? In many reinforcement learning problems, we need a balance of exploitation and exploration, and people are actively researching this tradeoff.

These kinds of problems can get even more complicated if we add different kinds of rewards, like a +1 battery and a +3 bigger battery. Or there could even be Negative Rewards that *Airor* needs to learn to avoid, like the black hole. If we let *Airor* explore this new environment using reinforcement learning, sometimes he falls into the black hole. So, the cells will end up having different values than the earlier environment, and there could be a different best policy. Plus, the whole environment could change in many of these problems.

If we have an AI in our car helping us drive home, the same road will have different people, bicycles, cars, and black holes on it every day. There might even be construction that completely reroutes us. This is where reinforcement learning problems get more fun, but much harder. When *Airor* was learning how to navigate on that small grid, cells closer to the battery had higher values than those far away. But for many problems, we'll want to use a **value function** to think about what we've done so far, and decide on the nex move using math.

For example, in situation where an AI is helping us drive home, if we're optimizing safety and we see the brake lights of the car in front of us, it's probably time to slow down, but if we saw a bag of donuts in the street, we would want to stop. So, reinforcement learning is a powerful tool that's been around for decades, but a lot of problems need a ton of data and a ton of time to solve. There have been really impressive results recently thanks to deep reinforcement learning on large-scale computing.

These systems can explore massive environments and a huge number of states, leading to results like AIs learning to play games. At the core of a lot of these problems are discrete symbols, like a command for forward or the squares on a game board, so how to reason and plan in these spaces is a key part of AI. Next, we'll dive into symbolic AI and how it's a powerful tool for systems we use every day.

Symbolic AI

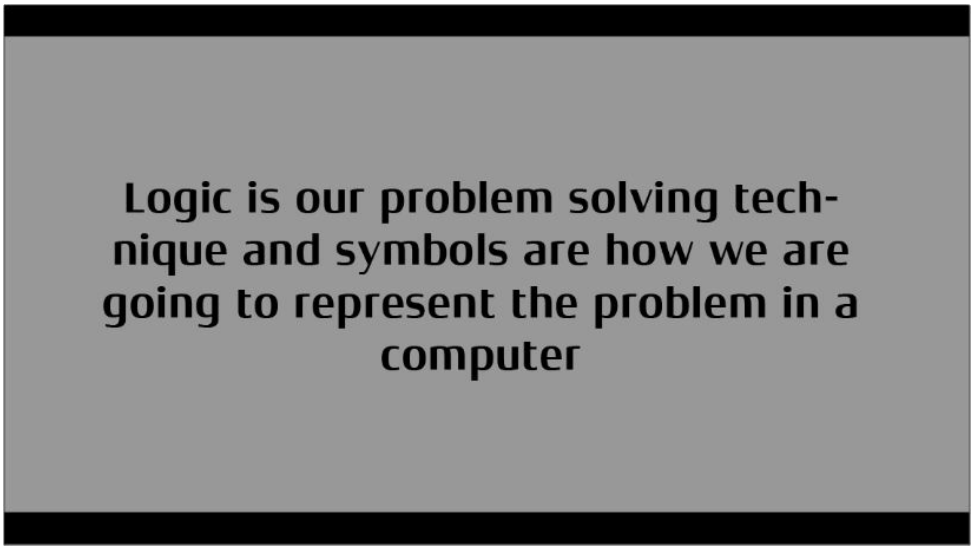
Back in 1959, three AI pioneers set out to build a computer program that simulated how a human thinks to solve problems. Allen Newell was a psychologist who was interested in simulating how humans think, and Herbert Simon was an economist, who later won the Nobel prize for showing that humans aren't all that good at thinking. They teamed up with Cliff Shaw, who was a programmer at the RAND corporation, to build a program called the General Problem Solver. To keep things simple, Newell, Simon, and Shaw decided it was best to think about the content of a problem separately from the problem-solving technique. And that's a really important insight.

For example, my brain would use the same basic reasoning strategies to plan the best route to work, school, or wherever I need to go, no matter where I start. Computers are logical machines that use math to do calculations, so logic was an obvious choice for the General Problem Solver's problem-solving technique. Representing the problem itself was less straightforward. But Newell, Simon, and Shaw wanted to simulate humans, and human brains are really good at recognizing objects in the world around us. So, in a computer program, they represented real-world objects as symbols. That's where the term **Symbolic AI** comes from, and its how certain AI systems make decisions, generate plans, and appear to "think."

If you've ever applied for a credit card, purchased auto insurance, or played a computer game newer than something like PacMan, then you've interacted with an AI system that uses Symbolic AI. Modern neural networks train a model on lots of data and predict answers using best guesses and probabilities. But Symbolic AI, or "good old-fashioned AI" as it's sometimes called, is hugely different.

Symbolic AI requires no training, no massive amounts of data, and no guesswork. It represents problems using symbols and then uses logic to search for solutions, so all we have to do is represent the entire universe we

care about as symbols in a computer.



Symbols can be anything in the universe: numbers, letters, words, bagels, donuts, toasters. One way we can visualize this is by writing symbols surrounded by parentheses, like (donut) or (bagels). A relation can be an adjective that describes a symbol, and we write it in front of the symbol that's in parentheses. So, for example, if we wanted to represent a chocolate donut, we can write that as chocolate(donut).

Relations can also be verbs that describe how symbols interact with other symbols. So, for example, I can eat a donut, which we would write as eat(name, donut) because the relation describes how one symbol is related to the other. Or we could represent *Airor's* relation to me, using sidekick(Airor, name). A symbol can be part of lots of relations depending what we want our AI system to do, so we can write others like is(Airor, robot) or wears(Airor, polo). All of our examples we are using will include a max of two symbols for simplicity, but you can have any number of symbols described by one relation.

A simple way to remember the difference between symbols and relations is to think of symbols as nouns and relations as adjectives or verbs that describe how symbols play nicely together. This way of thinking about symbols and their relations lets us capture pieces of our universe in a way that computers can understand. And then they can use their superior logic

powers to help us solve problems. The collection of all true things about our universe is called a **knowledge base** , and we can use logic to carefully examine our knowledge bases in order to answer questions and discover new things with AI. This is basically how Siri works. Siri maintains a huge knowledge base of symbols, so when we ask her a question, she recognizes the nouns and verbs, turns the nouns into symbols and verbs into relations, and then looks for them in the knowledge base.

Let's try an example of converting a sentence into symbols and relations, and using logic to solve questions. Let's say that " *Airor* drives a smelly, old, car." I could represent this statement in a computer with the symbols *Airor* and car, and the relations drives, smelly, and old. Using logical connectives like AND and OR, we can combine these symbols to make sentences called propositions. And then, we can use a computer to figure out whether these propositions are true or not using the rules of propositional logic and a tool called a truth table. Propositional logic is basically a fancy name for Boolean Logic. And the truth table helps us decide what's true and what's not. So, in this example, if the car is actually smelly, and actually old, and if *Airor* actually drives the car... then the proposition, "Smelly car AND old car AND *Airor* drives the car." is true.

We can understand that sort of logic with our brains: if all three things are true, then the whole proposition is true. But for an AI to understand that, it needs to use some math. With a computer, we can think of a false relation as 0 and true relations as any number that's not 0. We can also think of ANDs as multiplication and ORs as addition. But let's look at what happens to the math if the car is not actually old. Again, our brains might be able to jump to the conclusion that if one of the three things isn't true, then the whole proposition must be false. But to do the math like an AI would, we can translate this proposition as true times false times true, which is 1 times 0 times 1. That equals 0, which means the whole proposition is false.

So that's the basics of how to solve propositions that involve AND. But what if we want to know if *Airor* drives a car and that the car is either smelly OR old? Like I mentioned earlier, OR can be translated as addition. So, using our math rules, we can fill out this new, bigger truth table. The proposition we're dealing with now is "Smelly car OR old car AND *Airor* drives the car." For the first row, this translates as (true plus true), then that result times true, which we calculate as (1 plus 1) times 1. That equals 2 times 1, which is 2,

which means the whole proposition is true! Remember, any answer that isn't 0 is true.

The second row translates as (true plus false), then that result times true, which we calculate as (1 plus 0) times 1. That equals 1 times 1, which is 1, which means the whole proposition is true again. And we can fill out the rest of the truth table the same way!

SMELLY (CAR)	OLD (CAR)	DRIVES (AIROR,CAR)	SMELLY (CAR) AND OLD (CAR) AND DRIVES (AIROR,CAR)	SMELLY (CAR) OR OLD (CAR) AND DRIVES (AIROR,CAR)
(TRUE 1	TRUE 1)	TRUE 1	TRUE 1	TRUE 2
(TRUE 1	FALSE 0)	TRUE 1	FALSE 0	TRUE 1
(FALSE 0	TRUE 1)	TRUE 1	FALSE 0	TRUE 1
(FALSE 0	FALSE 0)	TRUE 1	FALSE 0	FALSE 0

Another logical connective besides AND and OR, is NOT, which switches true things to false and false things to true. And there are a handful of other logical connectives that are based on ANDs, ORs, and NOTs. One of the most important ones is called implication, which connects two different propositions. Basically, what it means is that IF the left proposition is true, THEN the right proposition must also be true. Implications are also called

if/then statements. We make thousands of tiny if/then decisions every hour (like, for example, IF tired THEN take nap or IF hungry THEN eat snacks).

Modern Symbolic AI systems can simulate billions of if/then statements every second! To understand implications, how about we use a new example: IF I'm cold THEN I wear a jacket. This is saying that if I'm definitely cold then I must be wearing my jacket, but if I'm not cold, I can wear whatever I want. So, if cold is true and jacket is true, both sides of the implication are true. Even if I'm not cold and I wear my jacket, then the statement still holds up. Same if it I'm not cold and I decide to not wear my jacket. I can do whatever since I'm not cold. BUT if I am cold and I decide not to wear my jacket, then the statement no longer works. The implication is false.

COLD (ME)	WEAR (JACKET)	IF COLD(ME) THEN WEAR(JACKET)
TRUE	TRUE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	TRUE
TRUE	FALSE	FALSE

Simply put, An implication is true if the THEN-side is true or the IF-side is false. Using the basic rules of propositional logic, we can start building a knowledge base of all of the propositions that are true about our universe. After that knowledge base is built, we can use Symbolic AI to answer

questions and discover new things! So, for example, if I were to help Airor start building a knowledge base, I'd tell him a bunch of true propositions.

Oh *Airor?* Alright, you ready *Airor?*

Raghavendra Yadav(Author) is a person,

Airor is a machine,

Car is a machine,

Car is old,

Car is smelly,

Airor is not person,

Raghavendra Yadav(Author) isn't machine,

Toaster is a machine.

You getting all this *Airor?* Clearly, at this pace, *Airor* would never be able to build a knowledge base with all the possible relations and symbols that exist in the universe. There are just too many. Fortunately, computers are really good at solving logic problems. So, if we populate a knowledge base with some propositions, then a program can find new propositions that fit with the logic of the knowledge base without humans telling it every single one. This process of coming up with new propositions and checking whether they fit with the logic of a knowledge base is called **inference** .

For example, the knowledge base of a grocery store might have a proposition that sandwich implies Between(Meat, Bread), or "IF sandwich THEN between(meat, bread)". Meat and Bread are the symbols, and Between is the relation that defines them. So basically, this proposition is defining a sandwich as a symbol with meat between bread. Simple enough. There might be other rules in the grocery knowledge base. Like, for example, a hotdog also implies Between(Meat, Bread), or "IF hotdog THEN between(meat, bread)." Now, if the grocery store is having a sale on sandwiches, should the hot dogs also be on sale? Well, with inference, the grocery store's AI system can apply the following logic: because sandwiches and hotdogs are both symbols that imply meat between bread, then hot dogs are inferred to be sandwiches, and the discount applies!

Over the years, we've created knowledge bases for grocery stores, banks,

insurance companies, and other industries to make important decisions. These AI systems are called expert systems, because they basically replace an expert like an insurance agent or a loan officer. Symbolic AI expert systems have some advantages over other types of AI that we've talked about, like neural networks. First, a human expert can easily define and redefine the propositional logic in an expert system. If a bank wants to give out more loans, for example, then they can change propositions involving credit score or account balance rules in their AI's knowledge base. If a grocery store decides that they don't want to discount hotdogs during the sandwich-sale, then they might redefine what it means to be a sandwich or a hotdog.

Second, expert systems make conclusions based on logic and reason, not just trial-and-error guesses like a neural network.

Third, an expert system can explain its decisions by showing which parts were evaluated as true or false. A Symbolic AI can show a doctor why it chose one diagnosis over another or explain why an auto loan was denied. The hidden layers in a neural networks just can't do that… at least, not yet. This, so-called, "good old-fashioned AI" has been really helpful in situations where the rules are obvious and can be explicitly entered as symbols into a knowledge base. But this isn't always as easy as it sounds. How would you describe a hand-drawn number 2 as symbols in a number knowledge base? It's not that easy. Plus, lots of scenarios are not just true or false, the real world is fuzzy and uncertain. As we grow up, our brains learn intuition about these fuzzy things, and this kind of human-intuition is difficult or maybe impossible to program with symbols and propositional logic.

Finally, the universe is more than just a collection of symbols. The universe has time, and over time, facts change and actions have consequences. So, next time we'll talk about these actions and consequences, and how robots use Symbolic AI to plan out their jobs and interact with the world.

Robotics

Robotics is a broad topic, because it's the science of building a computer that moves and interacts with the world (or even beyond the world, in space). Some of the most exciting AIs are robots that move through the world with us, gathering data and taking actions! Robots can have wings to fly, fins to swim, wheels to drive, or legs to walk. And they can explore environments that humans can't even survive in. But, unlike humans, who can do many different things. Robots are built to perform specific tasks, with different requirements for hardware and for learning.

Curiosity is a pretty amazing robot who spent 7 years exploring Mars for us, but it wouldn't be able to build cars like industrial robots or clean your apartment like a Roomba. Robotics is such a huge topic that it's also part of Computer Science, Engineering, and other fields. In the field of AI, robotics is full of huge challenges. In some cases, what's easy for computers (like doing millions of computations per second) is hard for humans. But with robotics, what's easy for humans, like making sense of a bunch of diverse data and complex environments, is really hard for computers. Like, for example, in the reinforcement chapter, we talked about walking, and how hard it would be to precisely describe all the joints and small movements involved in a single step.

But if we're going to build robots to explore the stars or, get me a snack, we have to figure out all those details, from how to build an arm to how to use it to grab things. So, we're going to focus on three core problems in robotics: **Localization** , **Planning** and **Manipulation** . The most basic feature a robot is that it interacts with the world. To do that, it needs to know where it is (which is localization) and how to get somewhere else (which is planning). So, localization and planning go hand-in-hand. We humans do localization and planning all the time. Let's say you go to a new mall and you want to find some shoes. What do you do? You start to build a map of the mall in

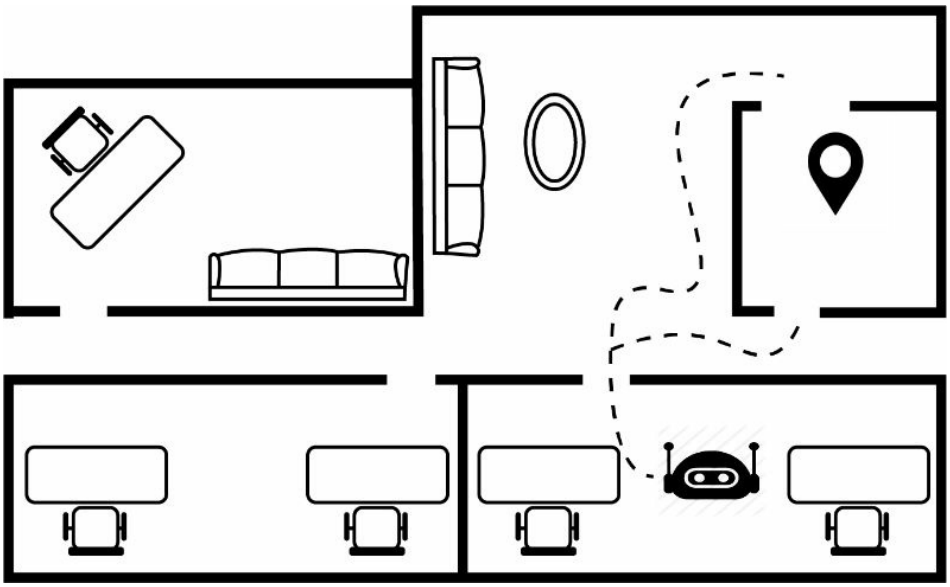
your head by looking around at all the walls, escalators, shops, and doors. As you move around, you can update your mental map and keep track of how you got there that's localization. And once you have a mental map and know the way to the shoe store, you can get there more quickly next time. So, maybe you can plan that the escalator is faster than the elevator. The most common way we input that data is with our eyes through perception. Our two slightly different views of the world allows us to see how far away objects are in space. This is called stereoscopic vision.

This mental map is the key to what many robots do too, if they need to move around the world. As they explore, they need to simultaneously track their position and update their mental map of what they see. This process is called **Simultaneous Localization and Mapping** , which goes by the cool nickname SLAM. But instead of eyes, robots use all kinds of different cameras. Many robots use RGB cameras for perception, which gather color images of the world. Some robots, use two cameras to achieve stereoscopic vision like us!

But robots can also have sensors to help them see the world in ways that humans can't. One example is infrared depth cameras. These cameras measure distances by shooting out infrared light (which is invisible to our eyes) and then seeing how long it takes to bounce back. Infrared depth cameras are how some video game motion sensors work, like how the Microsoft Kinect could figure out where a player is and how they're gesturing. This is also a part of how many self-driving cars work, using a technology called **LiDAR** , which emits over 100,000 laser pulses a second and measures when they bounce back. This generates a map of the world that marks out flat surfaces and the rough placement of 3D objects, like a streetlamp, a mailbox, or a tree on the side of the road. Once robots know how close or far away things are, they can build maps of what they think the world looks like, and navigate around objects more safely. With each observation and by keeping track of its own path, a robot can update its mental map. But just keep in mind, most environments change, and no sensor is perfect. So, a lot goes into localization, but after a robot learns about the world, it can plan paths to navigate through it. Planning is when an AI strings together a sequence of events to achieve some goal, and this is where robotics can tie into Symbolic AI from last chapter.

For example, let's say *Airor* had been trained to learn a map of the office,

and I wanted him to grab me a snack from the kitchen. He has localization covered, and now it's time to plan. To plan, we need to define actions, or things that *Airor* can do. Actions require preconditions, or how objects currently exist in the world. And actions have effects on those objects in the world to change how they exist. So, if *Airor's* mental map has a door between his current location and the kitchen, he might want to use an "open door" action to go through it. This action requires a precondition of the door being closed, and the effect is that the door will be open so that *Airor* can go through it. *Airor's* AI would need to consider different possible sequences of actions (including their preconditions and effects) to reason through all the routes to the kitchen in this building and choose one to take.



Searching through all these possibilities can be really challenging, and there are lots of different approaches we can use to help AIs plan. The third core problem of robotics: manipulation. What can *Airor's* mechanical parts actually do? Can he actually reach out his arms and interact with objects in the world? Many humans can become great at manipulating things (and I'm talking about objects, not that force powers stuff). Two traits that help us with manipulation, and can help our robots, are proprioception and closed loop control. Proprioception is how we know where our body is and how it's moving, even if we can't see our limbs.

Let's try an experiment: close your eyes, stretch your arms out wide, and point with both hands. Now, try to touch both index fingers without looking.

Our nervous system and muscles help our body's sense of proprioception. But most robots have motors and need sensors to figure out if their machine parts are moving and how quickly. The second piece of the puzzle is closed loop control or control with feedback. The "loop" we're talking about involves the sensors that perceive what's going on, and whatever mechanical pieces control what's going on. If you tried that experiment again with eyes open instead of closed, it would go even better. As your fingers get closer to each other, you can see their positions and make tiny adjustments.

I use my eyes to perceive, and I control my arms and fingers with my muscles, and there's a closed loop between them they're all part of my body and connected to my brain. It'd be a totally different problem if there was an open loop or control without feedback, like, for example, if you closed your eyes and tried to touch your finger to someone else's. Your brain can't perceive with their eyes or control their muscles, so you don't get any feedback and basically have to keep doing whatever I start doing. We use closed loop control in lots of situations without even thinking about it. If a box we're picking up is heavier than expected, we feel it pull the skin on our fingers or arms so we tighten our grip. If it's EVEN heavier than expected, we'd might try & involve our other hand, & if its too heavy, well, we'll call over my open-loop-example buddy.

But this process has to be programmed when it comes to building robots. Manipulation can look different depending on a robot's hardware and programming. But with enough work we can get robots to perform specific tasks like removing the cream from an oreo cookie. Beyond building capable robots that work on their own, we also have to consider how robots interact and coordinate with other robots and even humans. In fact, there's a whole field of Human-Robot Interaction that studies how to have robots work with or learn from humans. This means they have to understand our body and spoken language commands.

What's so exciting about Robotics is that it brings together every area of AI into one machine. And in the future, it could bring us super powers, help with disabilities, and even make the world a little more convenient by delivering snacks. But… we're still a long way from household robots that can do all these things. And when we're building and training robots, we're working in test spaces rather than the real world. For instance, a LOT of work gets done on self-driving car AI, before it even gets close to a real road. We

don’t want a flawed system to accidentally hurt humans. These test spaces for AI can be anything from warehouses, where robots can practice walking, to virtual mazes that can help an AI model learn to navigate. In fact, some of the common virtual test spaces are programmed for human entertainment: games. Next chapter, we’ll see how teaching AI to play games (even games like chess) can help us solve real-world problems.

AI Playing Games

AI researchers spend a lot of time trying to teach AI how to beat humans at games, & this isn't just because games are fun. Games provide constrained scenarios for testing new AI algorithms and approaches. In a game, it's easy to know whether the AI is winning or losing, because there's usually a score or some objective measure of "winning." This is great because AI learns from examples, trying things out, and slowly improving. Games basically provide their own training data, which is a big relief because AI systems need lots of training data to get really good.

An AI can even play against itself to generate training data and evolve better game strategies. Or an AI can be programmed to look at previous games (even games played by expert humans) for strategies that lead to victory. Comparing AIs against expert human gamers can also help us figure out how an AI is improving over time. This comparison also gives us a sense of the difficulty of problems an AI can solve. In other words, if I can teach *Airor* to beat me at battleship, I can probably also teach him to beat me at any game that's simpler than battleship.

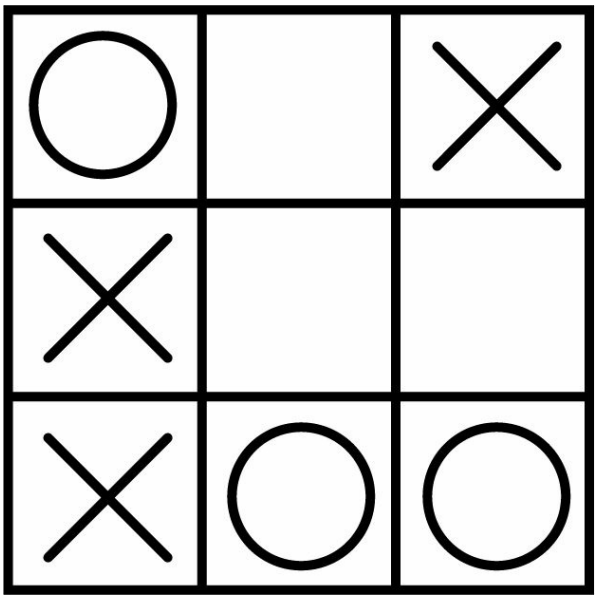
Finally, games are cool and that's important too! Sometimes AI programming can feel a bit difficult or dry because of all the math and troubleshooting, and games can provide a fun motivation to learn all this stuff. For all these reasons, games and computing have a rich history that's worth diving into.

Humans have always been fascinated by the idea that machines could beat us at our own strategy games. In 1770, the inventor Wolfgang von Kempelen revealed an invention to the Empress Maria Theresa of Austria. The Mechanical Turk was a clock-like contraption that appeared to play chess. Chess pieces were attached to rods that were hooked into a bathtub-sized box. After Empress Maria made a move on the board, she turned a crank that activated the machine, which would move chess pieces

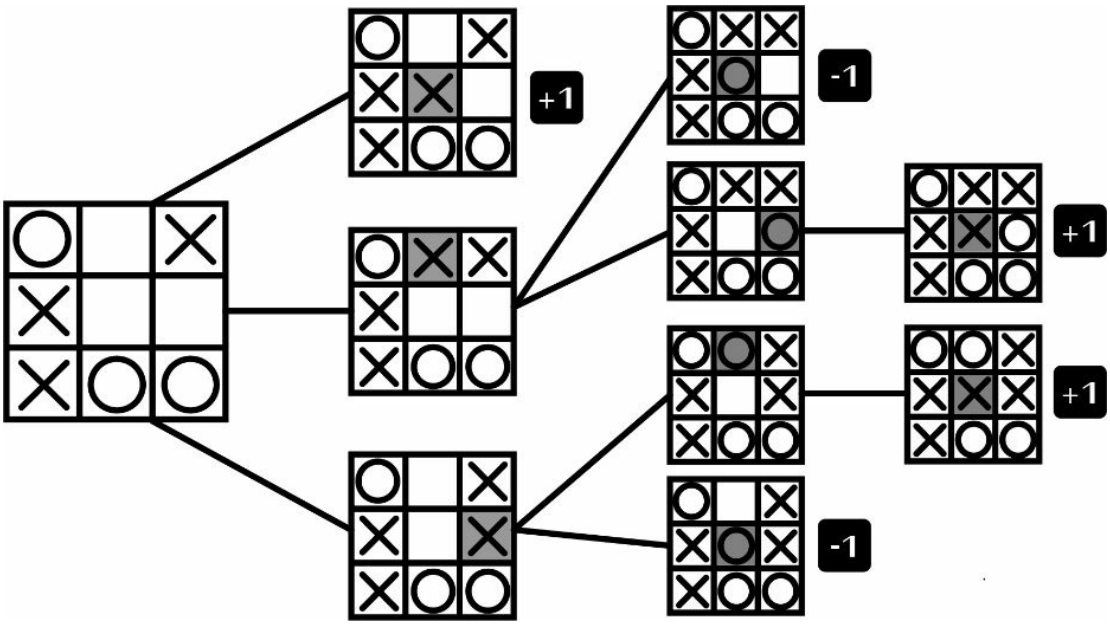
mechanically. To her surprise, the machine was able to beat most challengers. However, it was an elaborate hoax and The Mechanical Turk was actually controlled by a person hidden inside! Getting a machine to play chess is actually really complicated. So, when AI researchers first tried to tackle the problem in the late 1940s and early 1950s, they focused on simpler chess situations. Like, games with only a few pieces remaining on the board or full games played on a small 6x6 board without any bishops.

At the same time, researchers worked on an AI that could play checkers, because checkers looked easier… although it was really almost as complicated. The first program to play a legitimate game of checkers was built by IBM in 1956. And, in a classic cold war move, two programs that could play a full chess game were developed in parallel by the US and Russia in 1957. But these programs didn't get good for another 40 years. Checkers was first, with a program called Chinook which started dominating masters in 1995. Chess followed when a computer called Deep Blue beat the chessmaster Garry Kasparov in 1997. Since then, strategy games have been mastered one-by-one, with the most recent victories over humans at Go in 2017, DOTA 2 in 2018, and Starcraft II in 2019.

So, the best way to understand the difficulty of teaching AI to play games is through an example. Let's start with a really simple goal: teaching *Airor* to play tic-tac-toe. One of the ways that we can think about playing tic-tac-toe is as a tree with all the possible moves from any given state of what the game board looks like. For example



If this is the current game state, it's *Airor's* turn, and he's using Xs… there are three places he can go. We can draw a tree representing possible outcomes for each of these options, and all of the options his opponent (me, or anyone else) can take: Because computers think with numbers, each outcome can be assigned a reward a number like a 1 for a win, and -1 for a loss or tie. Basically, *Airor* will need to search through the tree of possibilities to find his win. To decide which choice to make, *Airor* will assume that in each tree, both he AND his opponent will make the best possible choices.

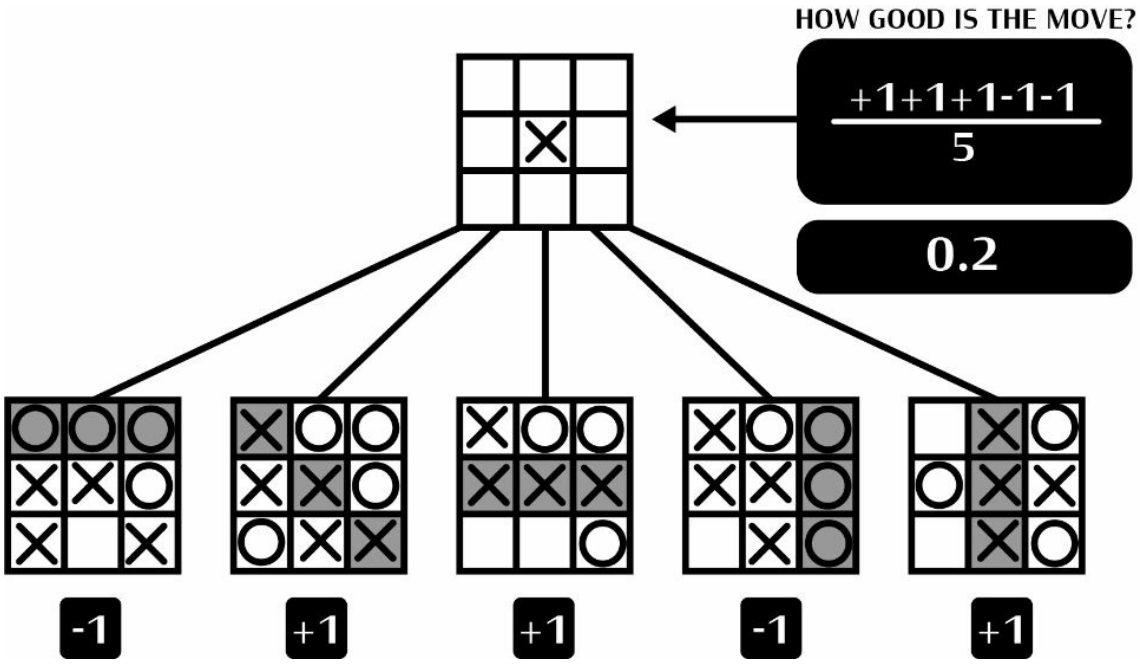


In other words, his policy (or his framework for making decisions) will alternate between choosing the branch that will maximize the outcome of winning on his turn, and minimize the outcome of his opponent winning on their turn. This is called the **minimax algorithm** . Then, each game state can be assigned a value based on how likely it leads to *Airor* winning, and he can decide which move to make based on his policy. Looking at this tree, *Airor* will always pick option 1.0, and win the game! Of course, this was a pretty small tree because we were looking at a game in progress. To draw the whole tic-tac-toe tree from beginning to end, we would need to represent about 250,000 boards. That seems like a lot, but it would take like a half a second for a powerful modern computer to compute this many options. By laying out all the possibilities and taking the paths that led to a win, *Airor* can solve tic-tac-toe. This means that *Airor* will always achieve the best possible outcome, either a win or a tie, no matter how his opponent plays.

But we can’t solve all games this way. Checkers, for example, has about 10 to the 20th power board states… or 10 followed by 20 zeros. That’s more board states than there are grains of sand on Earth! Chess has 10 to the 50th power board states. And Go has 10 to the 250th power board states. To put those huge numbers into perspective, there are only 10 to the 80th atoms in the entire known universe! Computer scientists have theorized that it would be impossible for conventional computers to calculate this many states due to

the laws of physics. Like, for example, if you combined all planets and stars and everything in the whole universe into a single supercomputer, it still wouldn't be powerful enough to solve the game of Go. But some people have hope that quantum computers may be able to get there one day... So, if figuring out all of the board states could be mathematically impossible, how did computers beat the number one ranked human masters in Chess and Go?

Many modern systems, including Google's AlphaGo computer that beat a human master in Go in 2017, use an algorithm called **Monte Carlo Tree Search** . Monte Carlo is a famous casino, so whenever you see the term "monte carlo," it's a good bet that the algorithm will be using randomness and chance to solve a problem. Combining Monte Carlo randomness and regular tree search like minimax, modern game AIs decide which part of the huge tree to search by guessing at odds. Basically, they want higher odds that the part of the game tree they search will lead to a win. But these aren't just random guesses like we would make in many casino games, AI systems can simulate millions of "what-if" scenarios and use math to estimate the likelihood of winning if they choose one path or another. In each "what-if" scenario, the AI considers making one particular move and then simulates playing a large number of (but not all) possible games, where the next moves are chosen at random. By averaging these possible outcomes, the AI estimates how "good" that particular move is.



It’s so much faster to estimate a handful of choices than exhaustively calculate each branch of the game tree. And some computers can even do this estimation in real time. One example of this is Google’s DeepMind which defeated human professional players at Starcraft II in 2019 - where time is very critical. Of course, Starcraft II, Go, and Tic-Tac-Toe aren’t all the types of games that humans play. Other games require other strategies and have other computational challenges: IBM’s Watson question-answering system was able to beat human Jeopardy! champions in two televised matches in 2011. Watson listened for keywords in the clue and tried to use a knowledge graph to figure out responses. And we’ll talk more about knowledge graphs later. Watson wasn’t perfect and struggled a bit with context. For example, it famously guessed “What is Toronto?” on something in the category “US Cities.” But Watson was still able to do better than human contestants overall.

Evolutionary neural networks use the environment as an input, like reinforcement learning. But this approach introduces multiple agents who try multiple neural network structures, and then build on successful ones for the next generation. Sort of like animals, the ones that are better at surviving get to pass on their genes. For example, the AI MarI/O can learn how to play a Super Mario World level by telling MarI/O what buttons it can push and that getting farther to the “right” in the level is good. This AI will start by

basically mashing buttons at random, but as some button mashes get it farther to the right, it remembers and learns from those successful attempts.

So, are there any games that are safe to play, where humans will always have an edge and AI won't be able to beat us? Computers definitely seem to struggle with parts of language like humor, irony, metaphor, and wordplay. Computers also aren't great at understanding and predicting real people, who don't always act "optimally," so social games could be more difficult too. But AI systems are finding some success in bluffing games like online poker, so it's important not to underestimate them. Computers might also struggle with creativity or surprise, because there's not a really clear way to assign values to states. It's difficult to assign a number to "how creative" something is, compared to saying "go as far right as you can in the Mario level" or "achieve a winning state in a chess game."

So, considering all of that, maybe games like charades would be pretty stacked for a human victory. Or… what about pictionary? Hide-and-seek???

Humans and AI working together

It seems like every time I look at the news, there's a new article about how AI and automation is going to take everybody's jobs! There is a way for AI and humans to work together, besides competing for the same jobs, resources, and game championships. Human-AI teams can use our strengths to help each other, and collaborate to fill in each other's weaknesses. Together, we can make better diagnoses, brainstorm new inventions, or imagine a future where humans and robots are working side-by-side.

AIs and humans have skills that can complement each other, AI can be good at searching through lots of possibilities and making some intelligent guesses at which one to pick. Plus, AI systems are consistent, and won't make mistakes because they're tired or hungry, like I sometimes do. On the other hand, humans can be good at insight, creativity, and understanding the nuances of language and behavior. We've learned from living in the world and interacting with each other, so we're better than AI systems at interpreting social signals. There are many ways that AI could support us, but a big one is that AI could amplify our decision-making activities with the right information.

Humans and computers can play games against each other, but when they join forces, they basically become a superhero dynamic duo. In chess, this particular kind of Human-AI team game is called advanced chess, cyborg chess, or centaur chess! In centaur chess, the computer does what it's great at: it looks several moves ahead and estimates the most promising next moves. And the human does what they're great at: they choose among uncertain possibilities based on experience, intuition, and even what they know about the opponent. Usually, the computer program is in the driver's seat for the early game, which is relatively mindless because there aren't too many move choices. But humans step in to guide the strategy in the middle game when things get more complicated.

The first centaur chess tournament was held in 2007, with the winning team led by Anson Williams. Anson's team, Intagrand, consists of him and the programs written by the Computer Science and Math experts Yingheng Chen and Nelson Hernandez. And the cool thing about this team is that it's currently considered to be the best chess player in the world, among humans and AI! Intagrand was able to win 2-0 against chess grandmasters, even though none of the team members are grandmasters. And in 2014, multiple AIs played against multiple centaur chess teams in competitions. Pure AI won 42 games, while centaur teams won 53 games! It seems like Human-AI collaboration is working out for chess, so that means it could be promising in other parts of our lives too.

Games provide a great constrained environment to demonstrate the possibilities of AI and, in this case, human-AI collaboration. Chess victories might not seem that significant, but similar Human-AI collaboration can be applied to other problems. AI takes on the parts that require memorization, rote response, and following rules. Humans focus on aspects that require nuance, social understanding, and intuition. For one, AI could help humans make decisions when we're dealing with large amounts of complicated information.

When a doctor is trying to make a diagnosis, they try to use their medical experience and intuition to make sense of their patient's symptoms and all the published clinical data. AI could help wade through all that data and highlight the most probable diagnoses, so the doctor can focus their experience and intuition on choosing from those which is where they'll be most helpful. Second, AI could help when humans are trying to come up with new inventions or designs, like a new engineered structure. An AI could apply predetermined physical constraints, like, for example, how much something should weigh or how much force it should be able to withstand. This lets the human experts think about the most practical designs, and could spark new creative ideas. Third, AI could also support and scale-up interaction between people. It could save people from doing rote mental tasks, so that they have more time and energy to help. For example, in customer support, virtual assistants can help answer easy questions about checking on an order or starting a return… or at least that's the goal. If you've ever tried these systems, you know that they can sometimes fail in spectacular ways, leaving you mashing the 0 button on your phone to try and get a representative on the

line. And fourth, robots that are AI-enabled but guided by humans could give people more strength, endurance, or precision to do certain kinds of work. Some examples of this may be an exosuit worn by a construction worker, or a remotely-guided search and rescue robot. These devices still need some AI to apply the right amount of force or navigate effectively, but all of the real decision-making is done by humans.

There are entire research fields, like Human-Computer Interaction and Human-Robot Interaction, dedicated to investigating and building new AI, Machine Learning, and Robotics systems to complement and enhance human capabilities. But it's not just that humans can become more effective with AI, AI also needs humans to succeed! This whole series we've been talking about how we can program AI to help them exist and learn, but a lot of human-AI interaction is more subtle. You could've supported an AI without even recognizing it. First, humans can provide AI with meaning and labels, because we have so much more experience with living in the real world. For example, if you've ever edited Wikipedia, you've contributed to Wikipedia-based algorithms such as WikiBrain. Because Wikipedia puts articles into nesting structures (like how an elephant is a mammal) and because articles link each other, algorithms can use this structure to understand the meaning-based connection between topics.

In fact, when we interact with digital technology, whether it's posting content, giving something a thumbs up, following driving directions on a phone, or typing a text message, we're often providing training data to help make AI systems more effective. Without our data, there wouldn't be recommended YouTube videos, predictive text messages, or traffic data for the GPS to use in suggesting a route. But sometimes providing personal data can be a double-edged sword which we'll get to in the Algorithmic Bias chapter. Secondly, humans can also try to explain an AI system's predictions, outputs, and even possible mistakes to other humans, who aren't as familiar with AI.

As we mentioned in the artificial neural networks, the reasons for an AI producing particular results can be tough to understand. We can see what data go into the program and which results fit the data well, but it can be hard to know what the hidden layers are doing to get those results! For example, an algorithm might recommend denying a customer's loan request. So, a loan officer needs to be able to look at the input data and algorithm, and then

communicate what factors might've led to denial. Many European countries are now making it a legal right to receive these kinds of explanations. Third, human experts can also inspect algorithms for fairness to different kinds of people, rather than producing biased results. Finally, AI doesn't understand things like the potential consequences of its mistakes or the moral implications of its decisions. That's beyond the scope of its programming. It's a common Sci-Fi trope that an AI built to minimize suffering might choose to eliminate all life on Earth, because if there's no life, there's no suffering! That's why humans may want to moderate and filter AI actions in the world, so we can make sure they line up with societal values, morals, and thoughtful intentions.

The bottom line is that organic and artificial brains may be better together, and through this book, you could be on the way to becoming one of those experts that works on the helpfulness and fairness of AI systems. In this chapter, we didn't focus on explaining one specific algorithm or AI technology. Instead, it's more about where our world might be going from the AI revolution that's happening now… besides just "automation replacing jobs."

We should recognize what data humans are providing to algorithms. What would it mean if we could claim some credit for the ways that our data have allowed algorithms to change lives for the better? Or how do we claim more power in cases where data are being used in potentially harmful or problematic ways? Second, we should think about if and how our human jobs could be made easier by working with AI although it's bound to be complicated. For example, people had similar concerns with the spread of personal computing and tools like spreadsheets. And yes, spreadsheets automated many bookkeeping tasks, which put many people out of work. Even though some types of jobs were destroyed, new accounting jobs were created that involved human-computer collaboration.

Technology took over more of the rote math calculations, and humans focused on the more nuanced and client-facing aspects of accounting work. Even though this idea can get overblown in the mass media, AI and automation has and will take people's jobs. No question. And we don't want to downplay the impact that has on people's lives. But by understanding how AI works, what it's good at, and where it struggles, we can also find opportunities to work more effectively and to create new types of jobs that

involve collaboration. Machines can help us do things that we can’t do as well (or at all) by ourselves. Human-AI collaboration can help us narrow down complex decision trees and make better choices.

Human-Robot collaboration has the potential to give us super strength or resilience. Different kinds of AI will impact the world in powerful ways, but not without costs. So it’s up to us to decide which costs are worth it, how to minimize harm, and create a future we want to live in.

Web Search

There used to be a time when a group of friends at dinner could ask a question like “is a hot dog a sandwich?” and it would turn into a basic shouting match with lots of gesturing and hypothetical examples. But now, we have access to a lot of human knowledge in the palm of our hands… so our friends can look up memes and dictionary definitions and pictures of sandwiches to prove that none of them have a connected bun like hot dogs.

Search engines are a huge part of modern life. They help us access information, find directions to places, shop, and participate in sandwich arguments. But how does Google find answers to questions? How are Siri and Alexa so smart but also easily stumped? How did IBM’s Watson beat the best Jeopardy players in the world? Well, search engines are just AI systems that are getting better and better at helping us find what we’re looking for.

When we talk about search engines, we typically think about the AI systems online, like Google, Bing, Duck Duck Go and Ask Jeeves. But the basic ideas behind non-AI search engines have existed for centuries. Essentially, search engines gather data, create organization systems to sort that data, and find results to a question. For example, when you needed an answer to a question and couldn’t search online, you could go to the library! Libraries gather data in the form of books and newspapers that are stacked neatly on the shelves. Librarians have organization systems to help you find what you’re looking for. Knowing that magazines are on shelves by the water fountain, while kids books are on the second floor is a kind of organization. Plus, fiction books are sorted by the author’s last name, while nonfiction has the Dewey Decimal System, and so on. Once you (or the librarian) have the resources you need, you’ll be able to find results to your question!

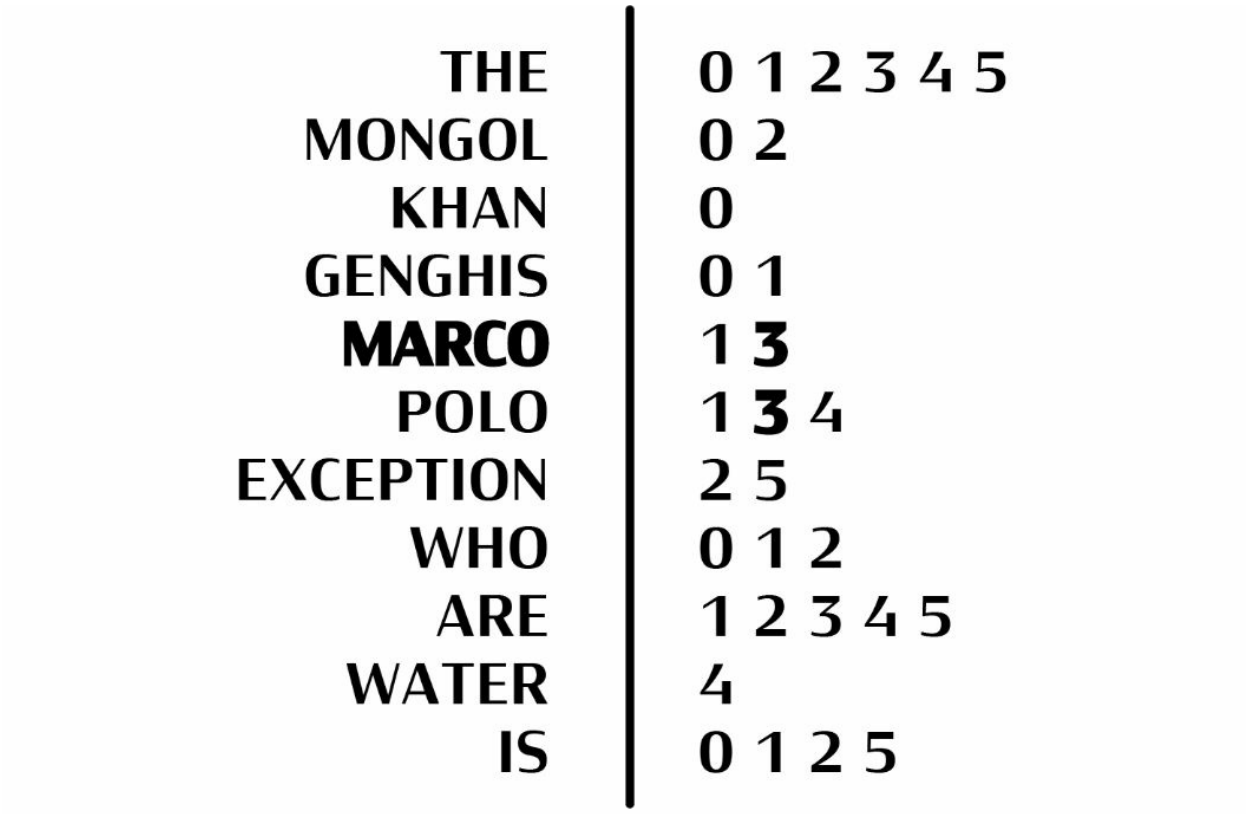
Now, rather than looking through books, web search engines look through all the data on the World Wide Web, aka “the Web”. And instead of asking a human librarian where to find information, we ask an AI like **Airor**

instead. And just so we're clear, we're using "Web" throughout this chapter even though it might sound a little old-fashioned. That's because the Internet and the Web are not the same thing.

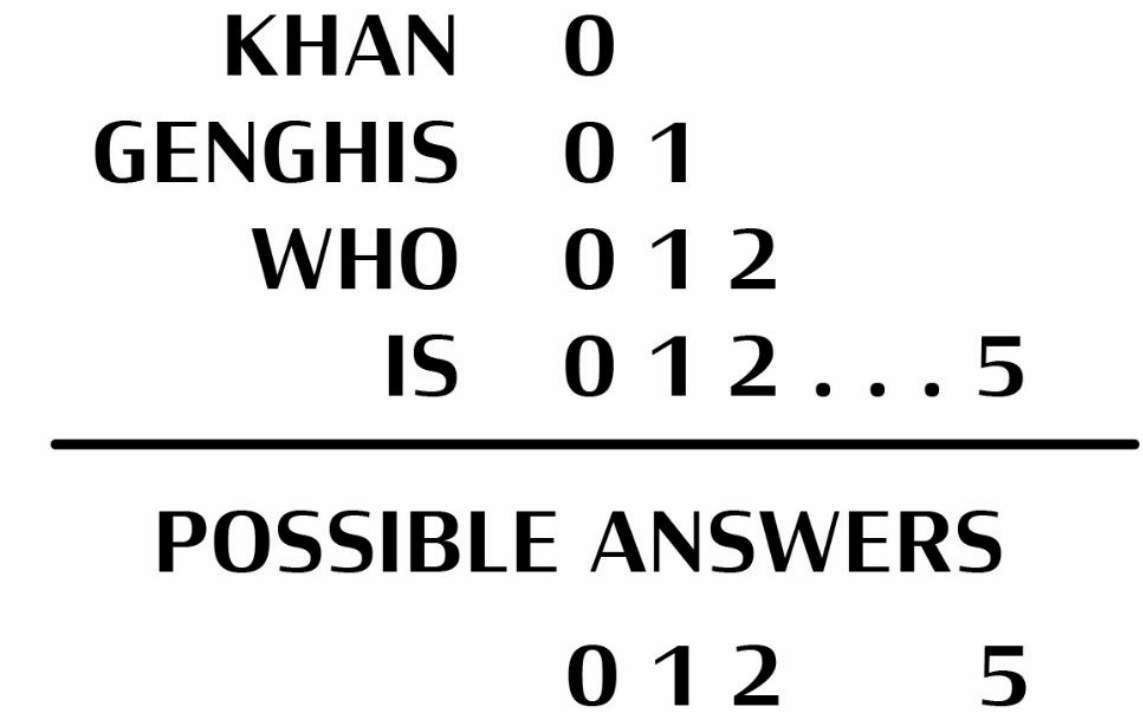
The Internet is a collection of computers that send messages to each other. Video services like Netflix that play on your TV, for example, use the Internet, not the Web. The Web, on the other hand, is part of the Internet and uses the Internet's connections to send documents and other content in a format that can be displayed by a browser like Chrome or Safari. As with most AI systems, the first step is to gather lots of data. To gather data on the Web, we can use a computer program called a **Web crawler** , which systematically finds and downloads Web pages. This is a HUGE task and happens before the search engine AI can take any questions.

It starts on some Web page that we pick, called a seed, and downloads that page and finds all its links. Then, the crawler downloads each of the linked Web pages and finds their links, and so on... until we've crawled the whole Web. After we have collected all the data, the AI's next step is to organize it by building an index, which is a kind of lookup system. The kind that's used for organizing Web pages is called an **inverted index** , which is like the index in the back of a textbook. For each word, it lists all of the Web pages that contain that word. Usually, the Web pages are represented by I.D. numbers so we don't have a long, messy list of URLs.

Let's say 0 is the seed - which happens to be a page about Genghis Khan. It has a lot of words on it like "the, mongol, Khan, Genghis, who, and is". In this inverted index, page 1 is about Marco Polo, but it mentions the word "Genghis" along with words like "the, Marco, Polo, who, are, and is." Page 2 is about the Mongols, page 3 is a different webpage about Marco Polo, and page 4 is about Water Polo.



So, let's say we type "Who is Genghis Khan?" into a search engine. Our AI can use this inverted index to find results, which in this case, are links to Web pages. The AI will look at the words "who", "is", "Genghis", and "Khan" and use the inverted index to find relevant pages. Our AI might find that Web pages zero, one, two and five have at least one of the words from the question "who is Genghis Khan?" When Siri says "I found this for you," the AI is just returning a list of Web pages that contain the same terms as the question.



Except… most search engines include one more step. There are millions of pages online that contain the same terms. So, it's important for search engines to rank Web pages, so that the top result is more likely to be relevant than the tenth result or the hundredth. Of course, Google and Bing don't hire "supervisors" to grade each possible question and answer to help their AI systems learn from training data. That would take forever, and they wouldn't be able to keep up with all the new content that gets created every day. Really, regular users like us do this training for free all the time. Every time we use a search engine, our behavior tells the AI whether or not the results answered our question. For example, if we type in "who is Genghis Khan" into a search engine, and click on a Web page about Star Trek II: The Wrath of Khan, we might be disappointed to find Genghis Khan isn't ANYWHERE in that movie. So, we'll bounce back to the search results, and try again until we find a page that answers our question. A bounce indicates a bad result. But if we click on a Wikipedia article about Genghis Khan and stay for a while reading, that's a click through, which probably means that we found what we were looking for… so that indicates a good result.

Human behaviour like bounces and click throughs give AI systems the training data they need to learn how to rank search results and better answer

our questions. Data from the Web and data from how we use the Web helps make better and better search engines. Now, sometimes we ask our smart devices questions and we want actual answers… not links to Web pages. When I say “OK Google, what’s the weather like in Indianapolis?” I don’t want to scroll through results. For this kind of problem, instead of using an inverted index, AIs rely on knowledge bases. Which you might remember from previous chapter about Symbolic AI.

A knowledge base encodes information about the universe as relationships between objects like "chocolate donut". One of the main problems with knowledge bases is that it’s really hard to write down all of the facts in the universe, especially common sense things that humans take for granted but computers need to be told. Enter AI researcher Tom Mitchell and his team of scientists from Carnegie Mellon University. In 2010, they created a huge knowledge base called the Never Ending Language Learner or NELL, which was able to extract hundreds of thousands of facts from random Web pages.

The way it works is really clever. NELL starts with some facts provided by a human, for example, the genre of music that Mozart plays is classical. Which was represented like this: Mozart. musicGenre. Classical. Similarly, Jimi Hendrix. plays. Guitar. And Darth Vader. hasChild. Luke Skywalker. Then, NELL gets to work and reads through each Web page one-by-one for words mentioned in those facts. Maybe it finds the text “Mozart plays the piano.” NELL doesn’t know much about these symbols, but this text matches the same pattern as one of the facts provided by a human, specifically, the “plays” relationship. So, NELL learns a new object: Piano. And a new fact: Mozart. plays. Piano. By searching over the entire Web, NELL can learn lots of facts based on just the three original ones that humans gave it! Some facts might appear hundreds or thousands of times online, like Lenny Kravitz. Has Child. Zoë Kravitz. But NELL might also find facts that are mentioned SOMEWHERE online and extract them as potentially true. Like, for example, Darth Vader. plays. Kloo Horn. We just don’t know!

Just like how we look for multiple sources when writing a paper, NELL uses repetition and multiple sources to build confidence that the facts it’s finding are actually true. To consider other relationships, NELL uses the highly confident facts it learned and searches through the Web again. Only this time, NELL is looking for new relationships. Maybe it finds the text

“Darth Vader cuts off Luke Skywalker’s hand,” and NELL learns a new (very specific) relationship: cutsOffHand. Over and over again, NELL will use known relationships to find new objects, and known objects to find new relationships creating a huge knowledge base.

AI systems can use huge knowledge bases, like this one extracted by NELL, to answer our questions directly. Instead of using the words from our questions to search through an inverted index, an AI like Siri can reformulate our questions into incomplete facts and then look for matches in a knowledge base. Using all these strategies, search engines have become really good at answering common questions. But questions like “How many trees are in Ohio?” or “How many hotdogs are eaten in the South Sandwich Islands annually?” still stump most AI systems, because not enough people ask them and AI hasn’t learned how to answer them well yet.

It’s also important to watch out for search engine answers to questions like “Who invented the time machine?” because AI systems have a tough time with nuance and incomplete data. And a big, sort of hidden, problem is that search engine AI systems, are influenced by any biases in data online. For example, if I ask Google for images of “nurses,” it will mostly show pictures of female nurses. So next time, we’ll talk about how an algorithm can be biased, where bias comes from, and what we can do to address bias in AI.

Algorithmic Bias and Fairness

Algorithms are just math and code, but algorithms are created by people and use our data, so biases that exist in the real world are mimicked or even exaggerated by AI systems. This idea is called algorithmic bias. Bias isn't inherently a terrible thing. Our brains try to take shortcuts by finding patterns in data. So, if you've only seen small, tiny dogs, you might see a Great Dane and be like "Whoa that dog is unnatural" This doesn't become a problem unless we don't acknowledge exceptions to patterns or unless we start treating certain groups of people unfairly. As a society, we have laws to prevent discrimination based on certain "protected classes" (like gender, race, or age) for things like employment or housing. So, it's important to be aware of the difference between bias, which we all have, and discrimination, which we can prevent. And knowing about algorithmic bias can help us steer clear of a future where AI are used in harmful, discriminatory ways.

There are at least 5 types of algorithmic bias we should pay attention to. First, training data can reflect hidden biases in society. For example, if an AI was trained on recent news articles or books, the word "nurse" is more likely to refer to a "woman," while the word "programmer" is more likely to refer to a "man." And you can see this happening with a Google image search: "nurse" shows mostly women, while "programmer" mostly shows mostly men. We can see how hidden biases in the data gets embedded in search engine AI. Of course, we know there are male nurses and female programmers and non-binary people doing both of these jobs! For example, an image search for "programmer 1960" shows a LOT more women. But AI algorithms aren't very good at recognizing cultural biases that might change over time, and they could even be spreading hidden biases to more human brains. it's also tempting to think that if we just don't collect or use training data that categorizes protected classes like race or gender, then our algorithms can't possibly discriminate. But, protected classes may emerge as correlated features, which are features that aren't explicitly in data but may be

unintentionally correlated to a specific prediction. For example, A controversial 2017 paper showed that sexual orientation is strongly correlated with characteristics of a social media profile photo.

Second, the training data may not have enough examples of each class, which can affect the accuracy of predictions. For example, many facial recognition AI algorithms are trained on data that includes way more examples of white peoples' faces than other races. One story that made the news a few years ago is a passport photo checker with an AI system to warn if the person in the photo had blinked. But the system had a lot of trouble with photos of people of Asian descent. Being asked to take a photo again and again would be really frustrating if you're just trying to renew your passport, which is already sort of a pain! Or, let's say, you got a cool gig programming a drone for IBM… but it has trouble recognizing your face because your skin's too dark… for example.

Third, it's hard to quantify certain features in training data. There are lots of things that are tough to describe with numbers. Like can you really rate a sibling relationship with a number? It's complicated! You love them, but you hate how messy they are, but you like cooking together, but you hate how your parents compare you... It's so hard to quantify all that! In many cases, we try to build AI to evaluate complicated qualities of data, but sometimes we have to settle for easily measurable shortcuts. One recent example is trying to use AI to grade writing on standardized tests like SATs and GREs with the goal to save human graders time. Good writing involves complex elements like clarity, structure, and creativity, but most of these qualities are hard to measure. So, instead, these AI focused on easier-to-measure elements like sentence length, vocabulary, and grammar, which don't fully represent good writing… and made these AIs easier to fool. Some students from MIT built a natural language program to create essays that made NO sense, but were rated highly by these grading algorithms. These AIs could also potentially be fooled by memorizing portions of "template" essays to influence the score, rather than actually writing a response to the prompt, all because of the training data that was used for these scoring AI.

Fourth, the algorithm could influence the data that it gets, creating a positive feedback loop. A positive feedback loop basically means "amplifying what happened in the past"… whether or not this amplification is good. An example is PredPol's drug crime prediction algorithm, which has

been in use since 2012 in many large cities including LA and Chicago. PredPol was trained on data that was heavily biased by past housing segregation and past cases of police bias. So, it would more frequently send police to certain neighborhoods where a lot of racial minority folks lived. Arrests in those neighborhoods increased, that arrest data was fed back into the algorithm, and the AI would predict more future drug arrests in those neighborhoods and send the police there again. Even though there might be crime in neighborhoods where police weren't being sent by this AI, because there weren't any arrests in those neighborhoods, data about them wasn't fed back into the algorithm. While algorithms like PredPol are still in use, to try and manage these feedback effects, there is currently more effort to monitor and adjust how they process data. So basically, this would be like a new principal who was hired to improve the average grades of a school, but he doesn't really care about the students who already have good grades. He creates a watchlist of students who have really bad grades and checks up on them every week, and he ignores the students who keep up with good grades. If any of the students on his watchlist don't do their homework that week, they get punished. But all of the students NOT on his watchlist can slack on their homework, and get away with it based on "what happened in the past." This is essentially what's happening with PredPol, and you can be the judge if you believe it's fair or not.

Finally, a group of people may mess with training data on purpose. For example, in 2014, Microsoft released a chatbot named Xiaoice in China. People could chat with Xiaoice so it would learn how to speak naturally on a variety of topics from these conversations. It worked great, and Xiaoice had over 40 million conversations with no incidents. In 2016, Microsoft tried the same thing in the U.S. by releasing the Twitterbot Tay. Tay trained on direct conversation threads on Twitter, and by playing games with users where they could get it to repeat what they were saying. In 12 hours after its release, after a "coordinated attack by a subset of people" who biased its data set, Tay started posting violent, sexist, anti-semitic, and racist Tweets. This kind of manipulation is usually framed as "joking" or "trolling," but the fact that AI can be manipulated means we should take algorithmic predictions with a grain of salt. This is why I don't leave *Airor* alone online…

The common theme of algorithmic bias is that AI systems are trying to make good predictions, but they make mistakes. Some of these mistakes may

be harmless or mildly inconvenient, but others may have significant consequences. To understand the key limitations of AI in our current society, Let's say there's an AI system called HireMe! that gives hiring recommendations to companies. HireMe is being used by Robots Weekly, a magazine where *Airor* applied for an editorial job. Just by chance, the last two people named "Ai" got fired from Robots Weekly and another three "Airs" didn't make it through the hiring process. So, when *Airor* applies for the job, HireMe! predicts that he's only 24% likely to be employed by the company in 3 years. Seeing this prediction, the hiring manager at Robots Weekly rejects *Airor* , and this data gets added to the HireMe! AI system. *Airor* is just another "Ai" that got rejected, even though he may have been the perfect robot for the job! Now, future "Ais" have an even lower chance to be hired. It's a positive feedback loop, with some pretty negative consequences for *Airor* . Of course, being named *"Airor"* isn't a protected class, but this could apply to other groups of people. Plus, even though algorithms like HireMe! Are great at establishing a link between two kinds of data, they can't always clarify why they're making predictions.

For example, HireMe! may find that higher age is associated with lower knowledge of digital technologies, so the AI suggests hiring younger applicants. Not only is this illegally discriminating against the protected class of "age," but the implied link also might not be true. *Airor* may be almost 40, but he runs a robot blog and is active in online communities like Nerdfighteria! So, it's up to humans interacting with AI systems like HireMe! to pay attention to recommendations and make sure they're fair, or adjust the algorithms if not.

Monitoring AI for bias and discrimination sounds like a huge responsibility, so how can we do it? The first step is just understanding that algorithms will be biased. It's important to be critical about AI recommendations, instead of just accepting that "the computer said so." This is why transparency in algorithms is so important, which is the ability to examine inputs and outputs to understand why an algorithm is giving certain recommendations. But that's easier said than done when it comes to certain algorithms, like deep learning methods. Hidden layers can be tricky to interpret. Second, if we want to have less biased algorithms, we may need more training data on protected classes like race, gender, or age. Looking at an algorithm's recommendations for protected classes may be a good way to

check it for discrimination. This is kind of a double-edged sword, though. People who are part of protected classes may (understandably) be worried about handing over personal information. It may feel like a violation of privacy, or they might worry that algorithms will be misused to target rather than protect them.

Even if you aren't actively working on AI systems, knowing about these algorithms and staying informed about artificial intelligence are really important as we shape the future of this field. Anyone, including you, can advocate for more careful, critical interpretation of algorithmic outputs to help protect human rights. Some people are even advocating that algorithms should be clinically tested and scrutinized in the same way that medicines are. According to these opinions, we should know if there are "side effects" before integrating AI in our daily lives. There's nothing like that in the works yet. But it took over 2400 years for the Hippocratic Oath to transform into current medical ethics guidelines. So, it may take some time for us to come up with the right set of practices.

Future of AI

We've covered a lot of ground together, from the basics of neural networks to game playing, language modelling, and algorithmic bias. And as we've been learning about different parts of artificial intelligence as a field, there have been a couple themes that keep coming up. First, AI is in more places than ever before. The machine learning professor Andrew Ng says that "Artificial Intelligence is the New Electricity." This is a pretty bold claim, but lots of governments are taking it seriously and planning to grow education, research, and development in AI. China's plan alone calls for over 100 billion U.S. dollars in funding over the next 10 years. Second, AI is awesome. It can help make our lives easier and sort of gives us superpowers. Who knows what we can accomplish with the help of machine learning and AI? And third, AI doesn't work that well yet. I still can't ask my phone or any "smart" device to do much, and we're far away from personal robot butlers. So what's next? What's the future of AI?

One way to think about the future of AI is to consider milestones AI hasn't reached yet. Current soccer robots aren't quite ready to take on human professionals, and Siri still has a lot of trouble understanding exactly what I'm saying. For every AI system, we can try and list what abilities would take the current technology to the next level. In 2014, for example, the Society of Automotive Engineers attempted to do just that for self-driving cars. They defined five levels of automation. For each additional level, they expected that the AI controlling the car can do more without human help.

At level 1, cruise control automatically accelerates and decelerates to keep the car at a constant speed, but everything else is on the human driver. At level 3, the car is basically on its own. It's driving, monitoring its surroundings, navigating, and so on... but a human driver will need to take over if something goes wrong, like really bad weather or a downed powerline. And at level 5, the human driver can just sit back, have a

smoothie, and watch any movie while the car takes them to work through rush-hour traffic. And obviously, we don't have cars with the technology to do all this yet. But these levels are a way to evaluate how far we've come, and how far our research still has to go. We can even think about other AIs using "levels of automation." Like, for example, maybe we have level 1 AI assistants right now that can set alarms for us, but we still need to double-check their work. But what are levels 2 through 5? What milestones would need to be achieved for AI to be as good as a human assistant? What would be milestones for computer vision or recommender systems or any of the other topics in this course?

Sometimes it's useful to think about different kinds of AI on their own as we make progress on each very difficult problem. But sometimes people try and imagine an ultimate AI for all applications: An Artificial General Intelligence or AGI. To understand why there's such an emphasis on being "general," it can be helpful to remember where all this AI stuff first started.

Alan Turing was a British mathematician who helped break the German enigma codes during World War II, and helped define the mathematical theory behind computers. In his paper "Computing Machinery and Intelligence" from 1950, he introduced the now-famous "Turing Test", or "The Imitation Game." Turing proposed an adaptation of a guessing game. In his version, there's an "interrogator" in one room, and a human and a machine in the other. The interrogator talks to the hidden players and tries to figure out which is a human and which is a machine. Turing even gave a series of talking points, like: Please write me a sonnet on the subject of the Forth Bridge. Add 34,957 and 70,764. Do you play chess? I have K at K1 and no other pieces. You have only K at K6 and R at R1. It's your move. What do you play?

The goal of The Imitation Game was to test a machine's intelligence about any human thing, from math to poetry. We wouldn't just judge how "real" a robot's fake human skin looks. As Turing put it: "We do not wish to penalize the machine for its inability to shine in beauty competitions, nor to penalise a man for losing in a race against an aeroplane." This idea suggests a unified goal for AI, an artificial general intelligence. But over the last 70 years, AI researchers focused on subfields like computer vision, knowledge representation, economic markets, planning, and so on. And even though we're not sure if an Artificial General Intelligence is possible many

communities are doing interdisciplinary research, and many AI researchers are taking baby steps to combine specialized subfields.

This involves projects like teaching a robot to understand language, or teaching an AI system that models the stock market to read the news and better understand market fluctuations. To be clear, most of AI is still science fiction… we're nowhere near Blade Runner, Her, or any similar movies. Before we get too excited about combining everything we've built to achieve AGI, we should remember that we still don't know how to make specialized AIs for most problems. Some subfields are making progress more quickly than others and we're seeing AI systems pop up in lots of places with awesome potential.

To understand how AI might be able to change our lives, AI Professors Yolanda Gil and Bart Selman put together the Computing Research Association's AI Roadmap for the next 20 years. They predict AI reducing healthcare costs, personalizing education, accelerating scientific discoveries, helping national defense, and more. Part of the reason they expect so much progress is that more people than ever (including us!) are learning how to build AI systems. And all of these problems have lots of data to train new algorithms. It used to be hard to collect training data, going to libraries to copy facts and transcribe books. But now, a lot of data is already digital. If you want to know what's happening on the other side of the planet, you can download newspapers or grab tweets from the Twitter API. Interested in hyperlocal weather prediction? You can combine free data from the weather service with personal weather stations to help know when to water your plants. And if you feed that data into a robot gardner, you could build a fully-automated weather-knowing plant-growing food-making garden!

Maker communities around the globe are combining data, AI, and cheap hardware to create the future and personalize AI technologies. While imagining an AI/human utopia is exciting, we have to be realistic, too. In many industries, automation doesn't only enhance human activities, it can replace humans entirely. Truck, delivery, and tractor drivers are some of the most common jobs in the US as of 2014. If self-driving vehicles revolutionize transportation in the near future, will all those people lose their jobs? We can't know for sure, but Gödel Prize winning Computer Science Professor Moshe Vardi points out that this is already the trend in some industries. For example, U.S. manufacturing output will likely keep rising, but

manufacturing jobs have been decreasing a lot. Plus, computers use energy, and that means we're not getting any benefits from AI for free. Massive amounts of machines running these algorithms can have a substantial carbon footprint. On top of that, as we've discussed, you have to be pretty careful when it comes to trusting AI systems because they often end up with all kinds of biases you may not want.

So, we have to consider the benefits of massive AI deployment with the costs. In a now-famous story from a few years ago, Target figured out a woman was pregnant based on her shopping history, and they sent her maternity coupons. But she was still in high school, so her family saw the mail, even though she hadn't told them. Do we want our data being used like this, and potentially revealing personal details? Or what about the government. Should it be allowed to track people with facial recognition installed on cameras at intersections? When we provide companies location data from our phones we could help them build better traffic models so we can get to places faster. Cities could improve bus routes, but it also means … someone … is … always … watching you. AI could also track your friends and family, where you shopped, ate, and who you hung out with.

If statistics have shown that people who leave home late at night are more likely to commit a crime... and an AI knows you left (even though it's just for some late night cookie dough), should it call the police to watch you just in case? So, we can go down any number of scary thought experiments. And there's a lot to consider when it comes to the future of AI. AI is a really new tool and it's great that so many people have access to it, but that also means there are very few laws or protections about what they can and can't do. Innovations in AI have awesome potential to make positive changes, but there are also plenty of risks, especially if the technology advances faster than the average person's understanding of it. It's probably the most accurate to say that the future is... complicated. And the most important thing we can do is be educated and involved in AI as the field changes. Which we're doing right now!

So, when we see a company or government rolling out a new technology, we know what questions to ask: Where did they get their data? Is this even a situation where we want AI to help humans? Is this the right tool to use? What privacy are we giving up for this cool new feature? Is anyone auditing this model? Is this AI really doing what the developers hoped it would?

GLOSSARY

Artificial intelligence	A subset of computer science that deals with computer systems performing tasks with similar, equal, or superior intelligence to that of a human.
AI winter	Artificial Intelligence Winters are periods of time during which artificial intelligence experienced reduced fundings for researches and low interest from the public.
Supervised learning	Training a model from input data and its corresponding labels.
Neural networks	Also known as artificial neural network, neural net, deep neuralnet; a computer system inspired by living brain.
Deep learning	Deep Learning is a subfield of machine learning concerned with algorithms that are inspired by the human

	brain that works in a hierarchical way.
Deep networks	A network architecture with many layers, typically 5–100.
Back propagation	An approach that is commonly used in the training process of the deep neural network to reduce errors.
Optimization	The process of setting decision variable values in such a way that the objective in question is optimized.
Learning rate	A scalar used to train a model via gradient descent.
Overfitting	Overfitting occurs when a model tries to predict a trend in data that is too noisy.
Unsupervised learning	Unsupervised learning is a type of machine learning algorithm used to draw inferences from sets of data consisting of input data without labeled responses,e.g., cluster analysis.
Unsupervised clustering	Immediately recognizing different properties and creating categories.
Representation	Set of techniques that

learning	allows a system to automatically discover the representations needed for feature detection or classification from raw data.
Auto encoder	A type of artificial neural network used to learn efficient data codings in an unsupervised manner.
Encoder-Decoder model	The encoder-decoder model is a way of using recurrent neural networks for sequence-to-sequence prediction problems.
Recurrent-neural network	A type of neural network that makes sense of sequential information and recognizes patterns, and creates outputs based on those calculations.
Reinforcement learning	A type of dynamic programming that trains algorithms using a system of reward and punishment.
Google Deep Mind	An artificial intelligence company founded in 2010 and later acquired by Google in 2014.
Credit assignment	The function that

	computes the value(s) used to update the weights.
Symbolic AI	Symbolic artificial intelligence is the term for the collection of all methods in artificial intelligence research that are based on high-level "symbolic" representations of problems, logic and search.
Knowledge based systems	It is a computer system that uses knowledge to solve a problem or support a decision.
Planning	A branch of AI dealing with planned sequences or strategies to be performed by an AI-powered machine.
Stereoscopic vision	The ability of the visual brain to register a sense of three-dimensional shape and form from visual inputs.
Simultaneous localization and mapping	The process of recording external environment of an AI machine like robot etc. SLAM is achieved by a combination of numerous sensors, receptors and emitters (for response).

LiDAR	The sensor which understands the world in 3D using laser beams and measures the time it takes for the laser to come back.
Closed loop control	Closed-loop control relies on sensory feedback that is usually as- summed to be free. But if sensing incurs a cost, it may be cost-effective to take sequences of actions in open-loop mode.
Minimax algorithm	A recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
Monte carlo tree search	It is a probabilistic and heuristic driven search algorithm that combines the classic tree search implementations alongside machine learning principles of reinforcement learning.
Web Crawler	A program or automated script which browses the World Wide Web in a methodical, automated

	manner.
Inverted index	An inverted index is an index data structure storing a mapping from content, such as words or numbers, to its locations in a document or a set of documents.